

Portland State University

PDXScholar

Dissertations and Theses

Dissertations and Theses

1987

Parallel approximate string matching applied to occluded object recognition

David Smith

Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Electrical and Computer Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Smith, David, "Parallel approximate string matching applied to occluded object recognition" (1987).

Dissertations and Theses. Paper 3724.


<https://doi.org/10.15760/etd.5608>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. For more information, please contact pdxscholar@pdx.edu.

AN ABSTRACT OF THE THESIS OF David Smith for the Master of Science in Electrical and Computer Engineering presented December 3, 1987.

Title: Parallel Approximate String Matching applied to Occluded Object Recognition.

APPROVED BY MEMBERS OF THE THESIS COMMITTEE:


Dr. F. Badi'i, Chairman


Dr. M. Perkowski


Dr. J. Riley


Dr. J. Semura

This thesis develops an algorithm for approximate string matching and applies it to the problem of partially occluded object recognition. The algorithm measures the similarity of differing strings by scanning for matching substrings between strings. The length and number of matching substrings determines the amount of similarity. A

classification algorithm is developed using the approximate string matching algorithm for the identification and classification of objects. A previously developed method of shape description is used for object representation.

The topics of approximate string matching and object recognition are discussed along with related research. The approximate string matching algorithm is explained in detail and experimental results from a simulation of the algorithm are presented. This thesis also proposes a hardware implementation of the algorithm, exploiting its parallel nature. The performance of the proposed architecture is discussed.

PARALLEL APPROXIMATE STRING MATCHING
APPLIED TO OCCLUDED OBJECT RECOGNITION

by
DAVID SMITH

A thesis submitted in partial fulfillment of the
requirements for the degree of


MASTER OF SCIENCE
in
ELECTRICAL AND COMPUTER ENGINEERING

Portland State University

1987


TO THE OFFICE OF GRADUATE STUDIES AND RESEARCH:

The members of the committee approve the thesis of David Smith presented December 3, 1987.


Dr. F. Badi'i, Chairman


Dr. M. Perkowski


Dr. J. Riley


Dr. J. Semura

APPROVED:


Dr. Lee Casperson, Head, Department of Electrical Engineering

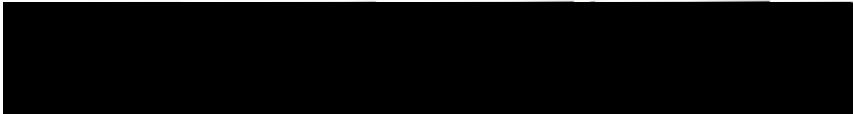

Dr. Bernard Ross, Dean, Graduate Studies and Research

TABLE OF CONTENTS

LIST OF TABLES	iv
LIST OF FIGURES	v
CHAPTER I	1
INTRODUCTION	1
APPROXIMATE STRING MATCHING	2
OBJECT IDENTIFICATION	3
MOTIVATION AND FOCUS	8
CHAPTER II	12
RELATED RESEARCH	12
CHAPTER III	17
INSM ALGORITHM	17
CHAPTER IV	21
DETAIL OF ALGORITHM	21
COMPARE FUNCTION	23
PROBLEMS OF ALGORITHM	38
EXPERIMENTAL RESULTS	40
CHAPTER V	46
HARDWARE IMPLEMENTATION	46
CHAPTER VI	58
PERFORMANCE	58
CONCLUSION	60
REFERENCES	62
APPENDICES	80

LIST OF TABLES

TABLE I	Integer Codes of INSM Algorithm.	18
TABLE II	Experimental Results	42

LIST OF FIGURES

Fig. 1	Object Identification	4
Fig. 2	Matching Process	5
Fig. 3	Parallel Computation	6
Fig. 4	Example Image with Error and Occlusion	8
FIG. 5	19
FIG. 6	24
FIG. 7	26
FIG. 8	28
FIG. 9	29
FIG. 10	29
FIG. 11	32
FIG. 12	32
FIG. 13	34
FIG. 14	34
FIG. 15	35
FIG. 16	37
FIG. 17	39
FIG. 18	Coded Plane with Noise.	43
FIG. 19	Coded Plane with Noise.	44
FIG. 20	A Register Network	48
FIG. 21	Internal EXNOR Connections	49
FIG. 22	A and B registers	50
FIG. 23	Variable Submatch Block	51
FIG. 24	Block C	52

FIG. 25	Processor Diagram	53
FIG. 26	Submatch Analysis Block	54
FIG. 27	Matchback Register Block	56

CHAPTER I

INTRODUCTION

Approximate string matching attempts to gauge how similar two strings are. This thesis develops a method for the measurement of relative string similarity and applies it to the problem of occluded object recognition and classification. As a specific example, the particular problems of aircraft recognition and classification are looked at.

Object recognition is an important problem in pattern recognition and requires especially complex and time consuming operations. This thesis deals with the problem of object classification after the object images are transformed to a representation based on Invariant Numerical Shape Modeling. Invariant Numerical Shape Modeling is a previously developed method for shape description based on the outlines of a shape. The image of an object is acquired by a computer vision system and stored in a computer. The algorithm for Invariant Numerical Shape Modeling, (INSM), is applied to the stored image. A set of numbers arranged as a string describing the outline is returned by the algorithm. This thesis is not concerned with the INSM algorithm itself

or methods of shape description, only the representations of shapes as strings.

APPROXIMATE STRING MATCHING

The problem of approximate string matching can be described as follows:

$$s \in S^*$$

where s is a string and S^* is the set of all strings over an alphabet A . The task is to find a string t which approximately matches s and where;

$$t \in T$$

$$T \subseteq S^*$$

Depending on the context of the problem, there may be more than one string in the set T which are sufficiently like s . There are two definitions of approximate; "sufficiently like" and "different but similar". "Sufficiently like" means two strings are functionally equivalent and can be substituted for each other in all contexts. "Different but similar" is more difficult to define but is the meaning used in this thesis. Judgement of similarity is highly intuitive and depends on the context of the problem. For instance, take two strings of numbers:

2 9 6 4 3

2 1 5 7 8

One basis for similarity could be the numerical values. One other could be the number of matching digits. An exact amount of difference can be found between two strings or the most similar string can be found over a set of strings. The algorithm developed here finds the most similar string out of a large set of strings, without determining the exact differences between strings. A numerical value is returned but it is only a relative value, applicable only to a particular string s and set T of strings.

Many applications for approximate string matching have been in text processing and databases. A good review of algorithms and applications is in the paper by Hall and Dowling, (1). A book edited by Sankoff, (2) details additional research on the general topic of sequence comparison and approximate matching.

OBJECT IDENTIFICATION

Figure 1 illustrates the basic process of object identification.

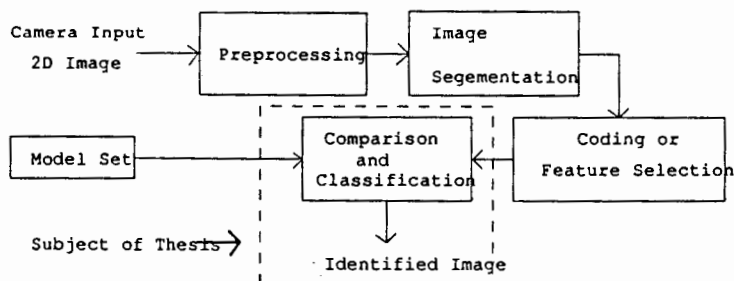


Figure 1.

Object Identification

A model set is established by taking pictures of all known objects. These objects are coded and organized into a library of image codes. This library is used by a classification algorithm or processor to identify unknown objects. An unknown object is acquired and processed by the same system used to capture the model set. The code is then input to the classification algorithm or processor.

The code is compared against each code in the library as illustrated in figure 2. If an unidentified code matches an identified code then the unknown image can be identified. However, as will be explained further on, it is almost impossible to obtain an exact match. The key feature of the algorithm developed here is that two codes do not have to match exactly. It allows for a wide margin of error and other differences. Although the identification may not be correct all the time, it will compute the most likely

identification. Figure 3 illustrates the parallel aspect of the process. The library can be divided up into parts, with each part going to a separate processor. Each processor computes the most likely identification for its set of library codes. A value is output indicating the relative similarity. The closest similarity match is found by taking the one with the highest value. Each processor operates in parallel with the others and all the processors are identical. The whole system is expandable because extra processors can be added up to a certain point and offer almost linear gains in total performance. The custom processor is discussed further in this paper and implements the algorithm developed here.

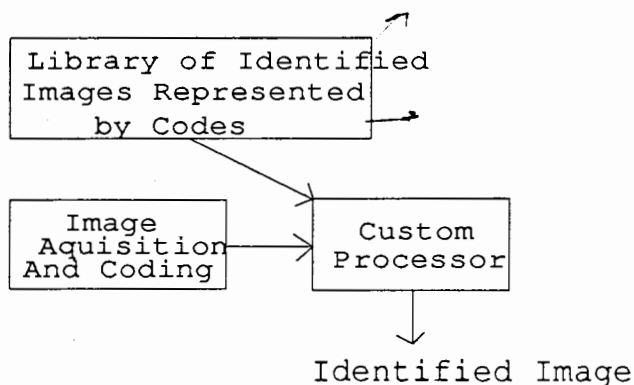


Figure 2.

Matching Process

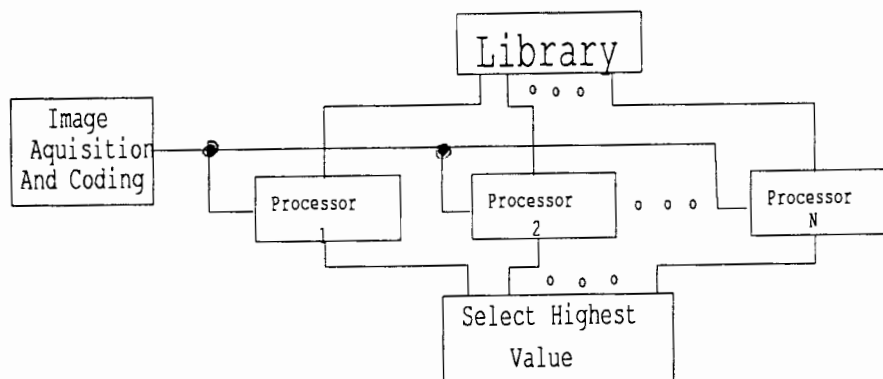


Figure 3.

Parallel Computation

Two of the problems of classification are errors and speed. Once a set of object models is established for a classification system, unidentified objects are compared to this set and identified. The classification system must be sensitive to the differences between classes in the set but ignore errors and distortions in the images. There are many causes of errors and distortions. Conditions during the identification process may not be the same as during the establishment of the model set. Lighting, distance, camera variations, and object positions can influence the final computer representation of an object. In addition to the aforementioned problems, other objects can come between the object in question and the camera, blocking out parts of the

image. The condition which best describes this is called occluding of the image. Therefore, part of the image may be missing or changed and it is impossible to exactly match an unidentified object to a known model.

Particular attention is given in this thesis to the problem of visually identifying aircraft. If an aircraft is to be identified by its image, the image must pass through several stages from the camera lens to its final representation in binary form. Every stage has the opportunity to introduce errors in the image and its binary representation. The errors may consist of slight aberrations in the outline of an image, slight differences in the geometry of an image, or other things. The final representation of the image can be quite different from what it should be. Every bit of error makes it harder to identify the object and increases the possibility of an incorrect identification.

Occlusions of the aircraft image can often occur. If the initial picture is taken from a distance, other objects can be in the way, partially occluding the image. These can be such things as clouds or other aircraft. It is also possible that an aircraft could be damaged, changing its appearance. These factors also make it hard to properly identify an image. Figure 4a shows an example of an original, correct image, while figure 4b shows the same image with an error or an occlusion present.

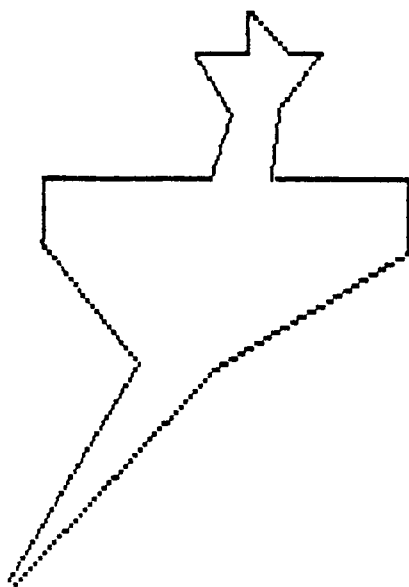


Figure 4a.

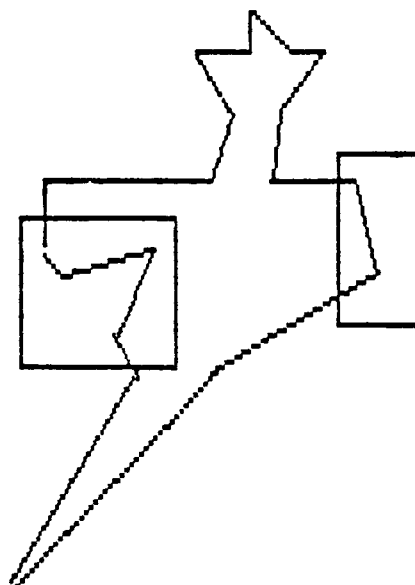


Figure 4b.

Example Image with Error and Occlusion

No matter what kind of algorithm is used to solve these problems, it must be very fast. The process of identification may require hundreds or thousands of comparison operations. Obviously identifying an aircraft should only take less than a second of real time, especially if a critical decision must be made. What has been developed in this thesis is an algorithm that attempts to solve these problems.

MOTIVATION AND FOCUS

The motivation for the development of the approximate string matching algorithm was the unique problem of occluded

object recognition using INSM codes. As with any algorithm for occluded object recognition, the key to similarity seems to be how much of an object matches the model object. When considering a segmented outline based representation, this equates to how many segments of the outline match the model. Because INSM codes use strings of numbers or symbols to represent objects, methods of approximate string matching seemed to be well suited for application.

The problem is that there may be multiple series' of symbols matching the object, separated by other series' of unknown length. Two strings cannot be compared simply on a one-to-one basis. The major limiting factor was discussed above, that of the speed of the algorithm. Reasonable accuracy is also expected.

The main focus of the research has been the development of an algorithm for the measurement of string similarity or for approximate string matching. This algorithm is applied to the problem of object recognition as a specific example. The two main goals in order were speed and accuracy.

There may be other algorithms which are more accurate but they are slow due to computational complexity. The key to speed is simplicity and the idea behind the algorithm is actually quite simple. The algorithm exploits the information made available from substring matching. The

properties of substring matching in parallel have apparently been overlooked by other researchers.

The algorithm is a rough attempt to model the process that a human being might use in trying to judge the similarity of two strings. The algorithm searches for matching substrings between two strings. If a human looks at two strings of symbols and asked to judge how similar they are, the human will see substrings. The position of the substrings is not critical up to a certain amount of displacement. Longer substrings will be more apparent than shorter ones and given more weight than the shorter ones. This cognitive process can also be used as a model for the similarity judgments of objects, (3). The primary structures that a human looks at in an object are edges. The edges are grouped together into groups using Gestalt principles of organization. Objects which share a maximum amount of groups could be judged more similar. Smaller groups or elements can be grouped together into larger groups. Larger groups tend to be recognized and compared better than smaller groups.

Chapter II discusses previous approaches to occluded object recognition and approximate string matching. Chapter III discusses Invariant Numerical Shape Modeling and how the it is applied to the algorithm developed. In chapter IV the algorithm is presented in detail and experimental results from a computer simulation of the algorithm are presented. Chapter V discusses a parallel hardware implementation of

the algorithm. Chapter VI discusses performance.

CHAPTER II

RELATED RESEARCH

This thesis combines two areas of research. Object and shape recognition and approximate string matching. Within the area of object and shape recognition, the problem of objects or shapes containing occlusions or errors is considered. There are two main approaches to recognition: statistical and syntactic.

Statistical pattern recognition generally relies on finding certain features such as edges, corners, and other special features. Templates of these features are formed and compared with pre-determined templates of known objects. Probability functions are used to determine the probability that an object can be classified as a specific object. Statistical methods have been developed for identification of partially occluded or hidden objects. Turney in (4) uses a Bayesian operation on templates formed from edge contours. Other approaches are shown in papers by Bhanu (5), Duda (6), and Wallace (7).

Generally, statistical methods are not precise and often return false locations for objects. They are also very computationally intensive. However, some methods are complete systems. They directly process the digitized

picture, whereas other methods including the algorithm developed here process data obtained from a picture via other algorithms.

Syntactic methods of shape recognition have been developed by Fu (8,9) and extended to 3-D object recognition by Lin and Fu (10). Syntactic methods are based on formal languages and quite similar to methods used in compilers. There are two main parts to syntactic recognition, initial processing and identification. The steps involved in initial processing are establishing a model set, which is done by selecting object primitives and generating a model grammar. The steps involved in identifying unknown objects are image capture and preprocessing, image segmentation, visible primitive surface identification and structural analysis and comparison. The process of identification is similar to sentence parsing. The letters and other terminals of a sentence are analogous to the visible primitive surfaces of an object. Structural analysis and comparison is similar to the procedure of sentence diagramming. Because of the extensive research into formal languages, much previous research can be applied to the field of object recognition. General parsing algorithms that handle errors have been developed (11,12), and in the paper by You and Fu (13), Earleys algorithm, which is well known, has been applied to 2-D images.

The advantage of using syntactic methods is that the recognition process is very accurate and can handle errors with a great deal of accuracy. The disadvantages can be many depending on the application. Syntactic recognition is very slow because of the computational complexity. In (13), the time to accept or reject an object with an error is on the order of 1 to 8 minutes using a mainframe computer. This time must also be multiplied for each object in the model set to be checked. In any type of real time environment, this is clearly unacceptable. Also in any environment outside the lab, a mainframe computer is unsuitable.

The research related to the string matching problem here is mainly in one area. The research involves the Damerau-Levenshtein metric, which is best illustrated by the research of Wagner, (14,15). Wagner's research has been applied to the area of text processing.

Wagner's algorithm is used to solve the string-to-string correction problem. The string-to-string correction problem is the computation of the distance between two strings based on the number of single character edit operations required to make one string equal to the other. The edit operations allowed are insertion of a character, deletion of a character, and changing a character. The Damerau-Levenshtein metric establishes the basic operations for editing and edit distance.

Wagner's algorithm returns an exact value of the distance in $O(m * n)$. The algorithm is constrained further in that two strings must start at approximately the same character. If the two strings are reversed with respect to each other, the distance between them will be much greater.

While Wagner's algorithm has been improved, (16,17) and VLSI hardware has been proposed by Cheng and Fu (18), it appears none of these methods are suitable both in time complexity and in application to the problem put forth in this paper. Wagner's algorithm uses a matrix and requires a lot of memory space and involves much sequential computation.

Probability methods for approximate string matching, (19,20), are not applicable to the problem defined here because of the computational complexity and the fact that methods in this area are designed for strings which essentially have the same ordering.

Syntactic methods for approximate string matching have also been developed but are similar to syntactic methods for object recognition.

Although there has been much research into object recognition, shape description, and approximate string matching, there has been only one paper published combining these subjects. Tsai and Yu, (21), have proposed a structural approach to shape recognition using methods of approximate string matching. They use previously developed

algorithms to construct a string representing the segmented outline of an image. However, each element of the string is actually a structure containing the length and relative angle of each segment. Strings are compared using the algorithm developed by Wagner. They also introduce a new edit operation for merging two or more segments into one segment.

While the method may be more accurate, it is clearly slower than the method developed in this thesis. However, apart from the image processing it is designed to operate on a general purpose computer. The method suffers from the sequential bottleneck of Wagner's algorithm. The additional information contained in the strings increases the complexity of the edit distance calculations.

CHAPTER III

INSM ALGORITHM

Invariant Numerical Shape Modeling (INSM), an algorithm developed for 2-D pattern recognition by Badi'i and Peikari (22), and extended by Badi'i and Majd (23) for recognition of 3-D objects forms the basis of the algorithm presented here.

INSM has been shown to be suitable for classification of 3-D objects when these objects are projected onto a 2-D image (23). This method takes an outline of the image and transforms it into a number of segments. These segments are transformed into a numerical code by tracing the outline. The transformation method is based on the lengths of the segments and the relative angles of the segments to each other. The coding is done by starting at a corner and first determining the length of a segment and then the angle between the segment and the following segment is used to determine if the following segment is a left turn or a right turn in the tracing of the image. The length of the following segment is also measured to determine if it is longer, shorter, or equal to the preceding segment. Table 1 lists the codes for all conditions.

TABLE I
 INTEGER CODES FOR INSM ALGORITHM

Relationship of Segments	Integer Code
<hr/>	
Right Turn	
Shorter	1
Equal	2
Longer	3
Left Turn	
Shorter	4
Equal	5
Longer	6
Spike	(7)
Shorter	71
Equal	72
Longer	73

Following in figure 5 is a example of the algorithm using an image of a plane:

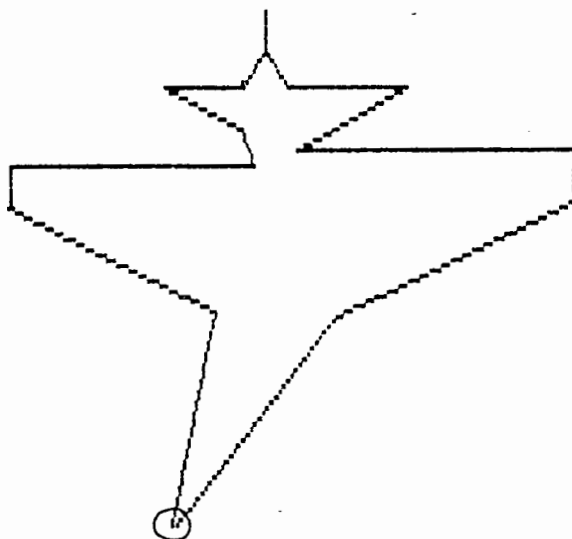


Figure 5.

Image of typical plane.

The coding is done from the corner circled:

left longer = 6	left shorter = 4
right shorter = 1	left longer = 6
right longer = 3	right longer = 3
left shorter = 4	left longer = 6
left longer = 6	right shorter = 1
right shorter = 1	right longer = 3
left shorter = 4	left equal = 5
left longer = 6	right shorter = 1
spike equal = 72	

The final code is 613461467246361351.

Using this method of shape description, numerical codes are generated for all possible orientations of a 3-D object using the 2-D image projection. It was shown by Badi'i in (23) that less than 100 images were needed to describe all possible views of an airplane, given a certain level of resolution. A library of codes for a range of plane models is set up and these codes are used to match and identify any unidentified planes. However, the INSM algorithm does not take into account any occlusions in the image or errors in the code. The basic purpose of the algorithm is to represent objects or shapes using a series of numbers. The representation is independent of the size and orientation of the shape or object. This thesis deals with the representations of objects as a series of numbers derived from the INSM algorithm.

CHAPTER IV

DETAIL OF ALGORITHM

The application of approximate string matching to object recognition requires the definition of a similarity function:

$$\sigma : S \times S \rightarrow R$$

Which for a pair of strings, (or INSM codes), produces a number $\sigma(s,t)$. The algorithm presented in this thesis accomplishes two things. It controls and organizes the process of classification given a model set and an unknown object code. The algorithm also proposes a definition for the similarity function.

The algorithm developed is a method of comparing two strings of numbers, a library code and an unidentified image code which may not be completely equal or the same length. Every code in the library of image codes is compared with the unidentified code and a match number is produced to indicate the approximate closeness or match of the two numbers. An unidentified image code is classified as a member of a class when the highest match number is obtained

when comparing the unidentified code and a library code of that class. This is shown by the following algorithm. The algorithm is applied separately to each library code along with the unidentified code.

```
HIGH_MATCH = 0
PLANE_TYPE = 0
FOR X = 1 TO MAX_LIBRARY_PLANE_CODES
  NEW_VAL = COMPARE(unident_code,library_codex)
  IF NEW_VAL > HIGH_MATCH THEN
    PLANE_TYPE = X
    HIGH_MATCH = NEW_VAL
  ENDIF
ENDFOR
```

Basically the value of the number returned by the algorithm or the function compare reflects the degree of closeness or matching on a relative basis. In this form of the algorithm, the HIGH_MATCH value does not equate to a specific similarity or distance value between the two codes. Because the INSM codes used are essentially strings, the algorithm can be considered as a method for approximate string matching.

This algorithm is designed to be implemented on a special purpose parallel processor. The parallelism exists

both in the algorithm as will be shown and in the parallel implementation of processors indicated in figure 3.

COMPARE FUNCTION

Following is a general explanation of how the compare procedure or similarity function operates. The difference function:

$$\sigma : S \times S \rightarrow R$$

has the following properties:

- i) $\sigma(s, t) \geq 0$
- ii) $\sigma(s, t) = [\text{length}(s)]^w$ iff $s=t$
- iii) $\sigma(s, t) = \sigma(t, s)$

The algorithm finds one string t which is most similar to string s for which $\sigma(s, t)$ is highest.

The following set of equations model the procedure:

$$\sigma(s, t) = \rho(s, t) + \rho(g(s), g(t))$$

where the function g is a mapping function:

$$g : X \rightarrow Y$$

$$Y_1 \leftarrow X_{(n/2)+1}, Y_2 \leftarrow X_{(n/2)+2}, \dots$$

$$Y_{(n/2)} \leftarrow X_1, \dots, Y_n \leftarrow X_{(n/2)}$$

In the above equation, n is the length of X .

$\rho(x,y)$ is described by the equation:

$$\rho(x,y) = \sum_{j=0}^{m-1} (\beta(h_j(x), y))$$

$h_j(x)$ is a rotational transposition function and j indicates the number of positions to be shifted, i.e. for $j = 1$;

$$X_1 \leftarrow X_n, X_2 \leftarrow X_1, \dots, X_n \leftarrow X_{n-1}$$

and m is the length of string x .

$$\beta(p,q) = \sum_{i=0}^k (f(\text{sub}_i(p,q)))^w$$

$$\text{where } f(\text{sub}(p,q)) = \begin{cases} 0, & \text{submatch length} < L \\ \text{sub}(p,q), & \text{submatch length} \geq L \end{cases}$$

and $\text{sub}(p,q)$ is a function that finds one substring match between strings p and q . The value k is the number of submatches which can be zero.

The value w is a number that controls the weight given to different length submatches. The value L is a number that controls the length of submatch accepted. The rationale for these two parameters is explained later.

$$\text{sub}(p,q) = \sum_{n=c}^d \text{comp}(p_n, q_n)$$

where c is the starting index of the submatch and d is the ending index.

$$\text{comp}(p_n, q_n) = \begin{cases} 1, & p_n = q_n \\ 0, & p_n \neq q_n \end{cases}$$

This set of equations forms the critical part of what has been developed in this thesis. However, the parallel nature of the algorithm may not be apparent but will be more clear in further explanations of algorithm. Additionally, heuristics have been formulated for application to the classification process and are explained later.

A more algorithmic explanation is formulated below. There are 4 basic tasks to the algorithm:

1. The comparison between the two codes.
2. The analysis of the results of the comparisons.
3. Parts 1 and 2 with a different orientation of the two codes.
4. Calculation of the number of digits successfully matched in each code.

Essentially, the algorithm returns a number called the match value. Each time the algorithm is applied to a library code and the unidentified code, a match value is obtained for the two codes. In other words, the unidentified code is

classified according to which library code has the highest match value. The match value is found by analyzing how many numbers or digits match between the two codes being compared. The algorithm looks specifically for substrings of digits in the two codes that match. The first part is to compare the two codes and find these substrings.

Figure 6 is an illustration of two different codes with three separate substrings that match. The length of each substring is greater than 2. From this point, the substrings will also be called submatches.

234	23	42123	3	43213421
234	42123	43213421		

Figure 6.

The value W indicated in the equations controls the weighting given to each matching substring. The basis for similarity measurement chosen is based on the longest common substring. There may be multiple matching substrings of different lengths. Many of them will be syntactically incorrect and out of sequence with the syntactically correct submatches. These incorrect submatches can contribute to a false value of relative similarity and an incorrect identification. However, based on observations of strings, these incorrect submatches are often short while the correct submatches are of longer length. By raising the length of a

submatch to the power w , longer submatches are given greater weight.

An important parameter of the algorithm is L , the length of submatches. $\mathcal{O}(s,t)$ is a function of the number and length of submatches. The selection of this value depends on the average length of strings being compared. It also depends to a lesser extent on the type of problem and the desired nature of the results. For the specific example of aircraft used in this thesis, $L = 3$ was selected. $L < 3$ results in more incorrect identifications. $L > 3$ results in fewer incorrect identifications but fewer correct ones as well. 3 was selected after analyzing the experimental results. In the following explanations and for the hardware implementation, $L = 3$ is assumed and the value 3 used directly.

```
IF LENGTH(submatch) < 3 THEN
```

```
    IGNORE submatch
```

```
ELSE
```

```
    ACCEPT submatch
```

Submatches of any length can occur but cases where

```
LENGTH(submatch) < 3
```

are quite numerous but affect the overall result very little while adding processing time. The majority of short submatches are incorrect and depend on the size of the alphabet. If the alphabet is small, then the probability of

short incorrect submatches is higher than if the alphabet size was larger. Therefore the heuristic of ignoring short submatches is used.

The comparison operation to find the submatches is done in parallel on each number in the two codes. This parallel operation is the key to the speed of the algorithm. A_i is compared directly with B_i , where A and B are the two codes and every comparison for each pair of numbers takes place at the same time. This is illustrated in figure 7 using the codes from figure 6.

A:	2	3	4	2	3	4	2	1	2	3	3	4	3	2	1	3	4	2	1
B:	2	3	4	4	2	1	2	3	4	3	2	1	3	4	2	1			

Figure 7.

If one of the codes is longer than the other then the two codes are left justified as indicated in figure 7. The extra numbers at the end of the longer code are ignored for the moment and only the numbers in each code having a one-to-one correspondence are compared. The results of a match are indicated by using a logical 0 to indicate that the two numbers are not equal and a logical 1 to indicate that the two numbers are equal. The results of each comparison are combined into groups of three, which allows us to disregard all submatches shorter than 3. This grouping is accomplished by comparing a 3 number substring for each number position in the shorter code. If there are 3

consecutive numbers matching in both codes then a 1 is written out to a code containing the results of the comparisons. Thus a 1 in this resultant code indicates a match of 3 consecutive numbers, two 1's indicate a match of 4 consecutive numbers and so forth. This can be shown by the following equation:

$$R_k = (A_k \odot B_k) * (A_{k+1} \odot B_{k+1}) * (A_{k+2} \odot B_{k+2})$$

for $k = 1, 2, 3, \dots, m - 2$

m equals the length of the shorter code

Where R represents a code containing the results of the comparison. The EXNOR sign implies a comparison operation that returns a logical 1 or logical 0. A and B represent the two codes being compared and k is the position in the resultant code. For clarity, note that each digit in A and B is actually represented by 3 bits and each bit of A and B must be compared separately. This is also shown in the hardware diagram of figure 20. The code R is analyzed to determine the number and length of submatches.

Again using the above codes as an example, this operation is illustrated in figure 8.

A:	2	3	4	2	3	4	2	1	2	3	3	4	3	2	1	3	4	2	1
B:	2	3	4	4	2	1	2	3	4	3	2	1	3	4	2				
R:	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0				

Figure 8.

For this orientation of these two codes only one submatch of 3 numbers is shown. The next operation is to shift the longer code one position to the right. This is a circular shift with the rightmost bit shifted into the leftmost position.

$$A_1 \leftarrow A_n, A_2 \leftarrow A_1, \dots, A_n \leftarrow A_{n-1}$$

Where n is the length of the longest code A .

This allows us to find all submatches greater than 2 regardless of their locations in the codes. We compare and circularly shift one position in the larger code until it has been completely rotated. At each step, the code R is analyzed. As an example if we compare and shift the above codes 3 times we have the orientation of the two codes as shown in figure 9.

A:	4	2	1	2	3	4	2	3	4	2	1	2	3	3	4	3	2	1	3
B:	2	3	4	4	2	1	2	3	4	3	2	1	3	4	2				
R:	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0				

Figure 9.

A submatch of 3 is indicated for the numbers 234 starting at the 7th position. If we intuitively evaluate the original codes, this submatch is actually false because it is not in the proper order with the other submatches. There is no way of avoiding these false matches but because correct submatches are longer and given more weight by the algorithm, the false matches do not contribute much to incorrect identifications. This method of weighting is explained further on in the paper. After 16 compares and shifts, we have the following codes as shown in figure 10.

A:	2	3	4	2	1	2	3	3	4	3	2	1	3	4	2	1	2	3	4
B:	2	3	4	4	2	1	2	3	4	3	2	1	3	4	2	1			
R:	1	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0			

Figure 10.

Here two submatches are indicated, one of length 3 and one of length 9. If we examine it more closely, referring to figure 6, we see the first submatch can be considered false and the second is actually one number longer than it really should be. Again, these inaccuracies do not affect the final answer very much.

For each compare and shift operation, we analyze the results in code R according to the lengths of all the submatches found. The match value is calculated by cubing the value of the length of each submatch.

$$\text{VALUE_OF_SUBMATCH}_i = \text{LENGTH}(\text{submatch}_i) ^ 3$$

$$\text{MATCH_VALUE} = \sum_{i=0}^K \text{VALUE_OF_SUBMATCH}_i$$

For each compare and shift operation a value is obtained and the values added up at each rotation step to obtain the final match value. For many of the steps or rotations of the longer code, there are no submatches and the evaluation step is skipped. The length of a submatch is found by looking at the lengths of consecutive strings of 1's in the result code R and adding the value 2. For example in figure 10 there are two strings of 1's in the result code R, one with a length of 1 and the other with a length of 7. This indicates submatches of length 3 and 9 between the two codes. Applying the portion of the algorithm as shown below:

$$756 = (1 + 2) ^ 3 + (7 + 2) ^ 3$$

The most value is given to those submatches that are the longest because of the cubing function. It turns out that although two codes may appear to differ by a large amount, there are enough long submatches between theoretical correct matches to provide correct identifications.

The third part of the algorithm rearranges the two codes and repeats parts 1 and 2, but keeping the match value

from the first run and adding to it during the second run of the 1st and 2nd parts. The two codes are rearranged according to the following formula:

$$A_1 \leftarrow A_{(n/2)+1}, A_2 \leftarrow A_{(n/2)+2}, \dots$$

$$A_{(n/2)} \leftarrow A_1, \dots, A_n \leftarrow A_{(n/2)}$$

$$B_1 \leftarrow B_{(n/2)+1}, B_2 \leftarrow B_{(n/2)+2}, \dots$$

$$B_{(n/2)} \leftarrow B_1, \dots, B_n \leftarrow B_{(n/2)}$$

The symbols n and m are the lengths of the respective codes.

The INSM codes are derived from the outline of an image, which of course has no beginning and end. The INSM code is generated from an arbitrary corner on the outline. The algorithm cannot find submatches which occur at opposite ends of the INSM codes. By cutting the two codes in the middle and bringing the ends together, this problem is solved. For example, look at the initial orientation of the following codes in figure 11.

A:	1	4	2	3	1	2	3	4	3	2
B:	1	3	1	4	2	3	4	3		
R:	0	0	0	0	0	0	0	0		

Figure 11.

Although some submatches will show up, the algorithm will miss a long submatch. Figure 12 illustrates this after rearranging the codes.

A:	2	3	4	3	2	1	4	2	3	1
B:	2	3	4	3	2	1	1	4		
R:	1	1	1	1	0	0	0	0		

Figure 12.

The final match value is the total of the match value found for the initial orientation of the two codes plus the match value found for the rearrangement of the two codes.

The fourth part of the algorithm is an important but separate part which calculates the number of digits in each code that are successfully matched at any comparison step. A code which is called the matchback code is generated for each of the INSM codes. Every position of the matchback code corresponds to a position in one of the INSM codes. For every compare and shift where a submatch is found, the results in code R are mapped back onto the matchback codes. For each digit pair successfully matched in every submatch of length greater than 2, the corresponding positions in the matchback codes are loaded with a logical 1. This is shown by the following equation:

$$A_MATCHBACK_K \leftarrow A_MATCHBACK_K + R_K$$

$$B_MATCHBACK_K \leftarrow B_MATCHBACK_K + R_K$$

In figure 13, using the codes from figure 8, the matchback codes are shown. A_MATCHBACK is on the top and corresponds to the top code A, while B_MATCHBACK is on the bottom and corresponds to code B. Since only one submatch of length three was found only those positions corresponding to the digit pairs matched have 1's. Note that where submatches of length 1 or 2 occur, no 1's are written to the matchback codes.

A_MATCHBACK:

	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
A:	2	3	4	2	3	4	2	1	2	3	3	4	3	2	1	3	4	2	1
B:	2	3	4	4	2	1	2	3	4	3	2	1	3	4	2				
	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0				

B_MATCHBACK:

R:	1	0	0	0	0	0	0	0	0	0	0	0	0	0					
----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	--	--	--

Figure 13.

Figure 14 shows the two codes A and B after 3 compare and shifts. Note that the values in A_MATCHBACK are shifted along with the code A. At this step in the algorithm, it can be seen 3 digits have been successfully matched in code B and 6 digits have been matched in code A.

	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	
A:	4	2	1	2	3	4	2	3	4	2	1	2	3	3	4	3	2	1	3
B:	2	3	4	4	2	1	2	3	4	3	2	1	3	4	2				
	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0				
R:	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0				

Figure 14.

After the code A has been completely shifted and the match value is found, the matchback codes are analyzed according to the following formula.

$$A_MATCHBACK_VAL = \text{NUMBER_OF_ONES}(A_MATCHBACK)$$

$$B_MATCHBACK_VAL = \text{NUMBER_OF_ONES}(B_MATCHBACK)$$

The function NUMBER_OF_ONES simply counts the number of logical ones in the code. A_MATCHBACK_VAL and B_MATCHBACK_VAL indicate how many digits were successfully matched in each code. Only one of the matchback values is used and the value is chosen according to the following formula:

$$MATCHBACK = \text{MINIMUM}(A_MATCHBACK_VAL, B_MATCHBACK_VAL)$$

Finally a percentage is calculated according to the following formula:

$$A_MATCH_PERCENTAGE = MATCHBACK / n$$

$$B_MATCH_PERCENTAGE = MATCHBACK / m$$

Where m and n are the lengths of the codes. Two percentages are calculated for each run of the algorithm on the initial and the rearranged codes. Although this information is not useful by itself, through further processing it provides an important heuristic to prevent incorrect identifications.

In some cases where one code was significantly longer than the other, an incorrect identification could occur. An example of this is in figure 15.

A:	2	3	1	4	2	3	1	4
B:	2	3	1	4				

Figure 15.

Because of the duplication of the B code in the A code, the final match value for this would be:

128 from the first run
+ 128 from the second run
 256 total

Assuming this is an incorrect identification, the percentage matched on the top code is only 50%. The heuristic is employed according to the following algorithmic statement:

```
IF( A_MATCH_PERCENTAGE-1 > CTV OR B_MATCH_PERCENTAGE-1 > CTV )  
AND ( A_MATCH_PERCENTAGE-2 > CTV OR B_MATCH_PERCENTAGE-2 > CTV )  
    ACCEPT match value  
ELSE  
    IGNORE match value
```

Where CTV contains a percentage that is preselected. This number varies with the average length of codes used and is also called the cutoff value since any match values with low percentages are cutoff. A specific mathematical relationship was not developed between the CTV and code lengths. By experimentation, the value of 65% was found to give optimum results for the experimental data discussed further on in this thesis.

PROBLEMS OF ALGORITHM

It must be noted this algorithm is not very precise, sacrificing speed for accuracy. This is due to unusual requirements of the basic problem. We have two codes, which are not equal, are probably of unequal length, have totally different rotations of numbers, and may differ from each other by an unknown amount. The errors and occlusions present in an unidentified code may be represented by one or more substrings of unknown length and of unknown placement in the unidentified code. There may also be missing parts of the code due to simplification of the image. The possible

combinations are almost infinite. It is also necessary that we analyze these two codes as fast as possible. Therefore there are tradeoffs between accuracy and speed which must be considered.

The main problems are false submatches and incorrect order of submatches. False submatches occur because there are several unusual combinations of numbers that can cause problems. Because of the rotational scheme, it is possible enough high submatches will show up in what is actually an incorrect total match. This is illustrated by figure 16:

A: |234|23|42123|3|4321342|1
 B: |4321342||42123||234|

Figure 16.

This results in a value of,

$$7^3 + 5^3 + 3^3 = 343 + 125 + 27 = 495$$

even though obviously by inspection the order is incorrect. False submatches occur when a substring in one code is replicated more than once in another. This is illustrated by figure 17:

A: |2345|31|2345|4|2345|
 B: |2345|21

Figure 17.

This results in a match value of,

$$4^3 + 4^3 + 4^3 = 64 + 64 + 64 = 192$$

Each occurrence of the substring in the bottom code is found in the top one. However, this example is an extreme case as the algorithm would throw the match value out because of the difference in bits matched per code. This feature of the algorithm is explained above.

The other problem we consider is the amount of errors present in short codes. If we take a short library code of 12 numbers and introduce one errors into the code, that error will change and/or add 3-5 numbers. Thus there is a greater chance of obtaining a higher match value for other library codes while getting a low value for the correct one because the amount of error is more than 25% to 40%.

EXPERIMENTAL RESULTS

A computer program was written in C to simulate the algorithm and to a limited extent parts of the hardware. A copy of the program is in Appendix 2. While the program is not meant to be a direct simulation of the the specialized hardware proposed for the algorithm, it successfully models the execution of the algorithm as it would be executed on the hardware level. However, there are parts of the program that do not simulate the algorithm completely. Sequential

operations must be used for the parallel comparison steps of the algorithm. The computer program served as a tool to test and modify the algorithm. Initial forms of the algorithm returned unsatisfactory results and the program was modified several times as the algorithm changed until it reached its present form.

A library was set up based on 5 airplanes described in (23). Drawings of a Phantom, a MIG, a Mirage, a B57, and a F105 were used as the library images. A total of 144 codes were used to represent all possible configurations of the planes. 10 airplane images representing different code lengths were selected for testing. Each of the images were altered by adding 1 or 2 errors, occlusions and/or structural modifications. The 38 resulting INSM codes were computed by hand and the resulting codes differed by one or two unequal segments of varying lengths from the original codes. 38 separate codes were obtained in this manner. After the planes were modified and codes generated, the plane images were drawn using computer drafting for presentation in the thesis. While pictures of all the planes used are shown in appendix 3, not all of the modifications are pictured.

Additionally, the average distance between the modified codes and the original codes was computed. For modified codes with one modification, the distance was an average 27 percent. For modified codes with two

modifications, the distance was an average 47 percent. The average length of the modified codes was 21 digits and the average length of the library codes was 16. It must be noted there is no exact way to measure the difference between two codes. The best way is to use the intuitive judgement of a human being.

Each one of these codes was input to the program and compared with the entire library. Appendix 4 contains the output of the program this set of data. The results are summarized in table II.

TABLE II
EXPERIMENTAL RESULTS

Number of changed segments	Percentage of correct matches	Percentage of incorrect matches	Percentage of no match indicated
1	91%	9%	0%
2	56%	31%	13%

A correct match is one where the highest value was returned for a comparison between a modified code and the original code and the plane was identified as the correct, intended model. An incorrect match is one where the plane identified was the wrong one. When a no match is indicated, then there were no matches meeting a certain percentage of numbers matched in the codes. It also may be possible to

correctly identify the class of plane while still incorrectly identifying the plane code. However, in this set of data, this case did not occur.

Following is an example of data used for the experiment. Shown in figure 18 is a plane and then the plane with an error added to the picture.

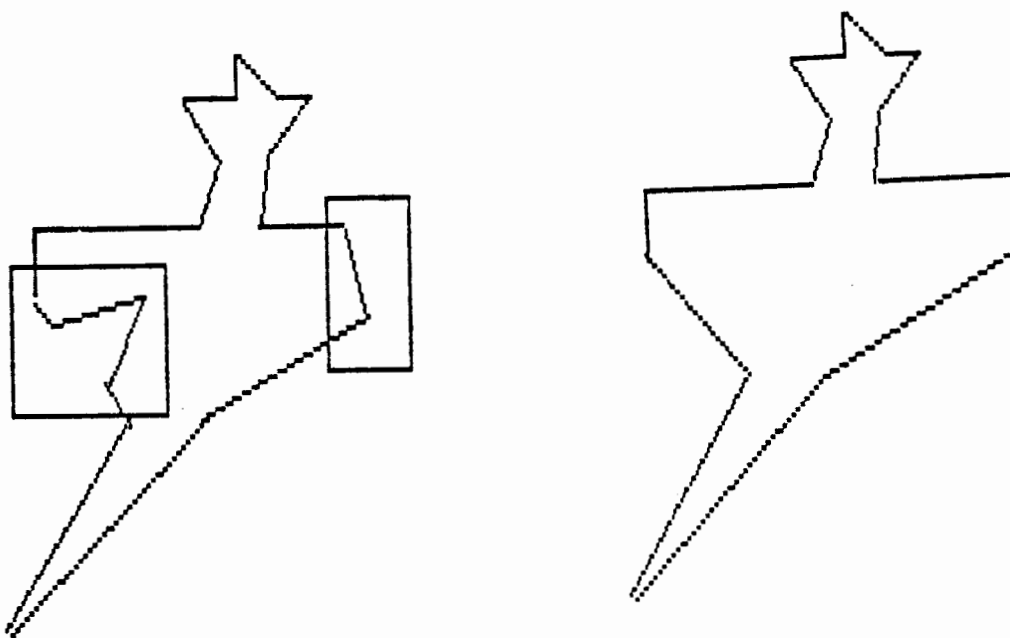


Figure 18.

The INSM code for the unmodified plane was:

6343461361413461

The INSM code for the modified plane was:

634346236413461

The match value returned for the modified plane's code when matched against the first code was 2342. The next highest value returned was 1334 for a plane very similar in design.

All the values for the other library codes were significantly lower.

Another example is an unmodified code of:

72613463426146136

And the modified code of:

72642614613672634616

The match value returned, correctly identified the modified code as the correct plane, even though the difference between the two is about 50%. This plane is shown in figure 19. Only segments 1 and 2 were used in this particular example.

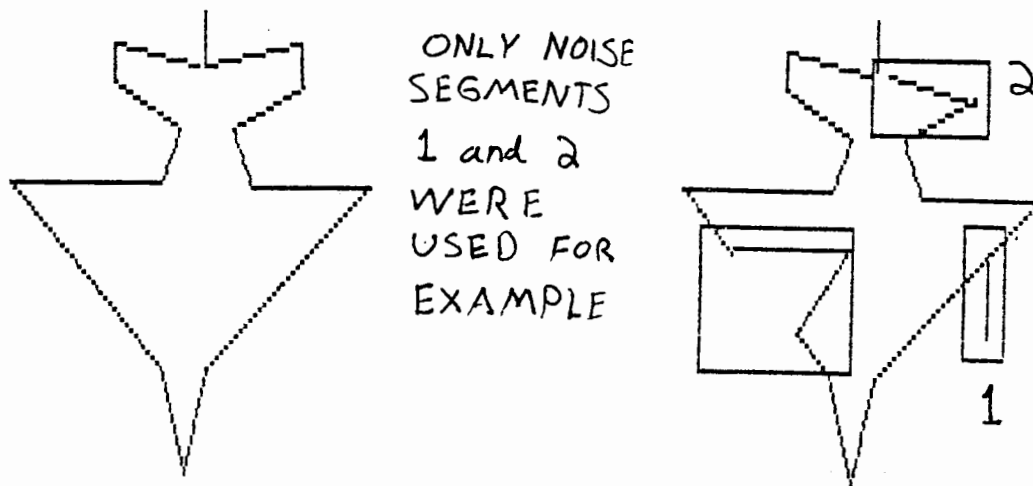


Figure 19.

The overall results indicated that independent of the number of erroneous segments, the algorithm worked for

modified codes differing from original ones by up to 30-50%. The algorithm was more successful when the original code was longer. When images with shorter original codes were used, just one minor change could cause a 50% change. With the average code length of 20 numbers the maximum number of changed areas is about 2. If longer image codes were used, the algorithm would work for images with more errors. However, it should be noted this increase may also cause an increase in errors. Another factor along these lines is increased library size. If the image codes are longer then the resolution will be higher, possibly requiring more images per plane to adequately describe all orientations. This increase may cause more incorrect identifications because there would be a greater incidence of false submatches.

CHAPTER V

HARDWARE IMPLEMENTATION

The algorithm was designed for a parallel hardware implementation. The hardware structures closely resemble the algorithmic or software structures. The operation of the hardware follows the algorithm exactly except for a few differences. The A and B registers and the matchback registers are represented by a two dimensional array in the algorithm in appendix 1. In order for the simulator to be written, certain hardware structures such as this are modeled by software structures. Another difference to be considered is that in the algorithm, each code is made up of digits. The value of each digit can range from 0 to 7 and can be represented by 3 bits. The hardware has to deal with the codes represented by bits so there is a level of hardware dealing with codes as 3 bit blocks. This is illustrated further on in the hardware description.

The image of the object targeted for identification is acquired by an image acquisition system and processed to obtain an outline of the image. The outline is processed by a program to obtain an INSM code. This is the code to be input to the processor and loaded into the A register. The library code is loaded into the B register and the index of

the library code is loaded into the Library Code Index register. The length of each code is determined by hardware in the registers and given to the microcontroller. Although previous examples in this thesis indicate register A would contain the longer code, this doesn't make any difference in the hardware implementation. The A and B registers have a special feature allowing any length of code to be circularly shifted. According to the algorithm, the codes in A and B must be circularly shifted, however the codes may be shorter than the register length. Therefore the register position representing the end of the code must be shifted to the front of the code. This is accomplished using a switch to select the correct register position. The details of the A and B registers are shown in figure 20. It must also be pointed out because each digit in each code is made up of three bits, each bit must be shifted in order and each shift actually represents 3 shifts of one bit each.

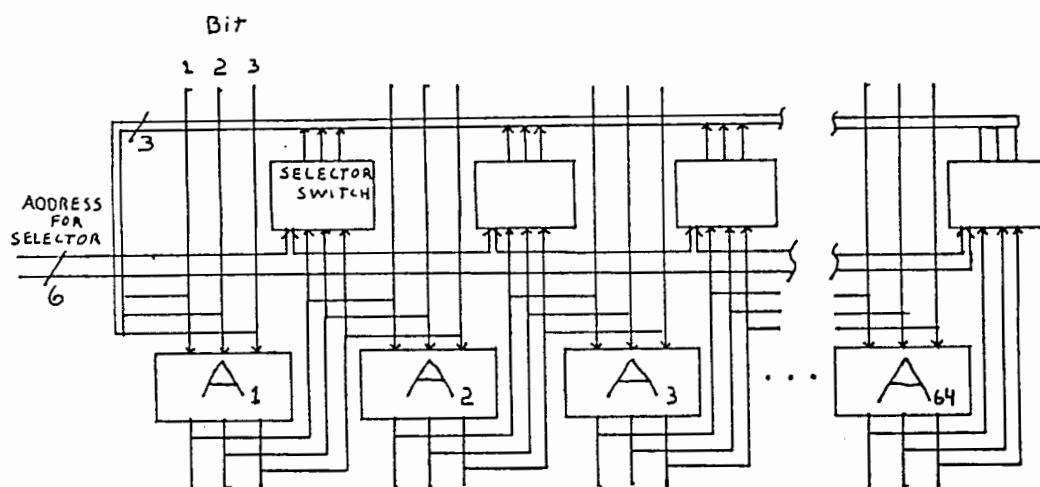


Figure 20.

A Register Network

The comparison of the codes is done using a EXNOR-AND network. In the diagram of figure 22, the first level of EXNOR gates are actually implemented using the gate network shown in figure 21. Because each code is represented by 3 bits, each bit position of the two codes must be compared separately thus we expand the following equation for comparison:

$$R_k = (A_k \odot B_k) * (A_{k+1} \odot B_{k+1}) * (A_{k+2} \odot B_{k+2})$$

for $k = 1, 2, 3, \dots, m - 2$

m equals the length of the shorter code

The resulting equation reflecting the bitwise representation is:

$R_k =$

$$\begin{aligned}
 & [(A_{k-1} \odot B_{k-1}) * (A_{k-2} \odot B_{k-2}) * (A_{k-3} \odot B_{k-3})] * \\
 & [(A_{k+1-1} \odot B_{k+1-1}) * (A_{k+1-2} \odot B_{k+1-2}) * (A_{k+1-3} \odot B_{k+1-3})] * \\
 & [(A_{k+2-1} \odot B_{k+2-1}) * (A_{k+2-2} \odot B_{k+2-2}) * (A_{k+2-3} \odot B_{k+2-3})] \\
 & \text{for } k = 1, 2, 3, \dots, m - 2
 \end{aligned}$$

m equals the length of the shorter code

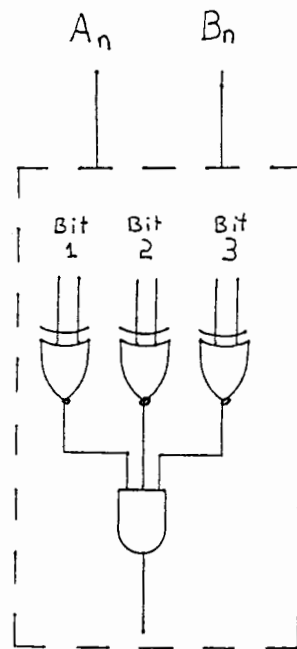


Figure 21.

Internal XNOR Connections

Once it is determined the bits of the two digits of the two codes are equal, then the comparison operation can continue with a single bit representing the result of that level of comparison. It is not necessary to deal with the information as blocks of 3 bits after that point in the algorithm.

The A and B registers are arranged as in figure 22, where the XNOR gates are as explained above. The output of the XNORs are input to the variable submatch detection hardware.

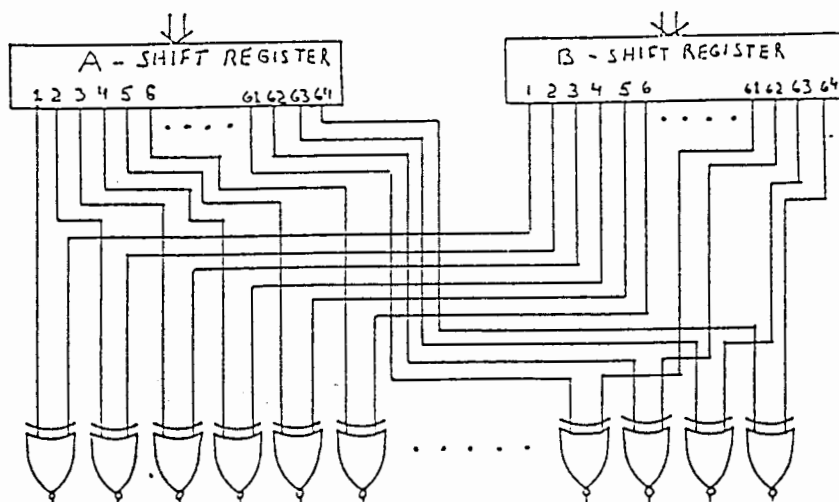


Figure 22.

A and B Registers

The variable submatch detection hardware is shown in figure 23. This allows the processor to select the parameter L or the minimum length submatch detected. L is selected by setting control signals S1, S2, S3, S4 to the left of the

figure. The hardware block assumes a minimum length of 2. The maximum value of L is 6. The value for L chosen depends on the type of data being processed. The following table lists the possible values of L and the corresponding control signal values.

<u>L</u>	<u>S1</u>	<u>S2</u>	<u>S3</u>	<u>S4</u>
2	1	1	1	1
3	0	1	1	1
4	0	0	1	1
5	0	0	0	1
6	0	0	0	0

Block C is shown in figure 24 and is identical for every AND gate.

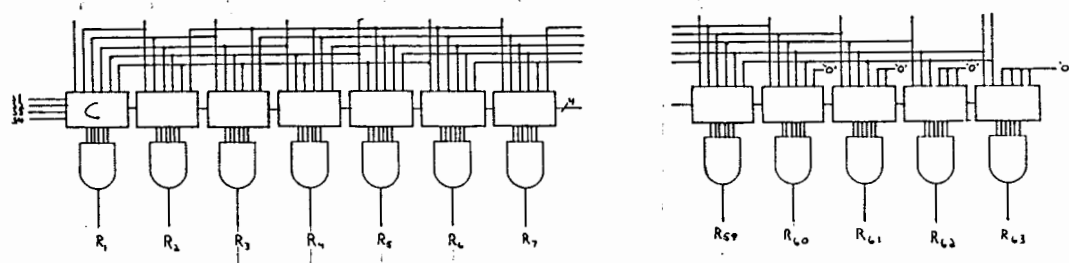


Figure 23.

Variable Submatch Block

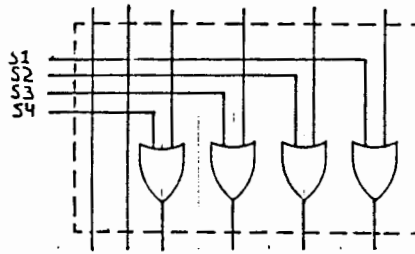


Figure 24.

Block C

The output of the variable submatch detection block is R from the algorithmic explanation. The value or code R is input to the submatch analysis hardware and to the matchback registers. A block diagram of the entire processor is shown in figure 25.

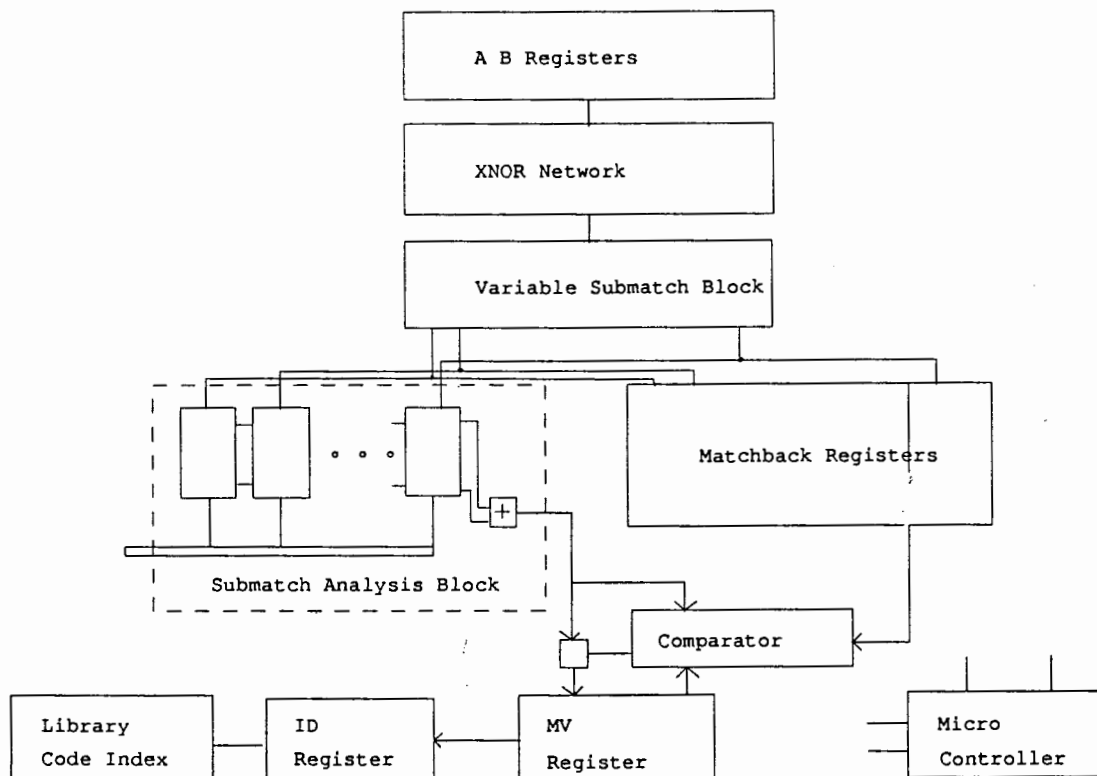


Figure 25.

Processor Diagram

The submatch analysis block is an iterative circuit which operates in blocks of 4 bits each. Each element of the iterative circuit is made up of random logic, adders, and memory. An element is shown in figure 26.

S1, S2, S3 control a correction factor related to the parameter L. The RAM holds the values for the W weighting function. This function raises the value of a submatch length to W. Rather than a complicated and time consuming operation to calculate the exponential function, a table-lookup method is used. The RAM is loaded with a set of values dependent on the parameter W. A non-exponential or non-linear set of values can also be chosen depending on the data being processed.

The circuit is designed to run asynchronously. The maximum propagation time is calculated. At the end of that time, the final answer is present at the output. A typical propagation time would be on the order of a few microseconds. This block will calculate a correct answer if there is one substring match or several substring matches.

The matchback block is shown in figure 27. The match back registers take the output of the R register. At the end of each run, the iterative counters add up the number of digits matched in each code and the minimum of the two is selected and loaded into the cutoff register to be compared with the preselected cutoff value.

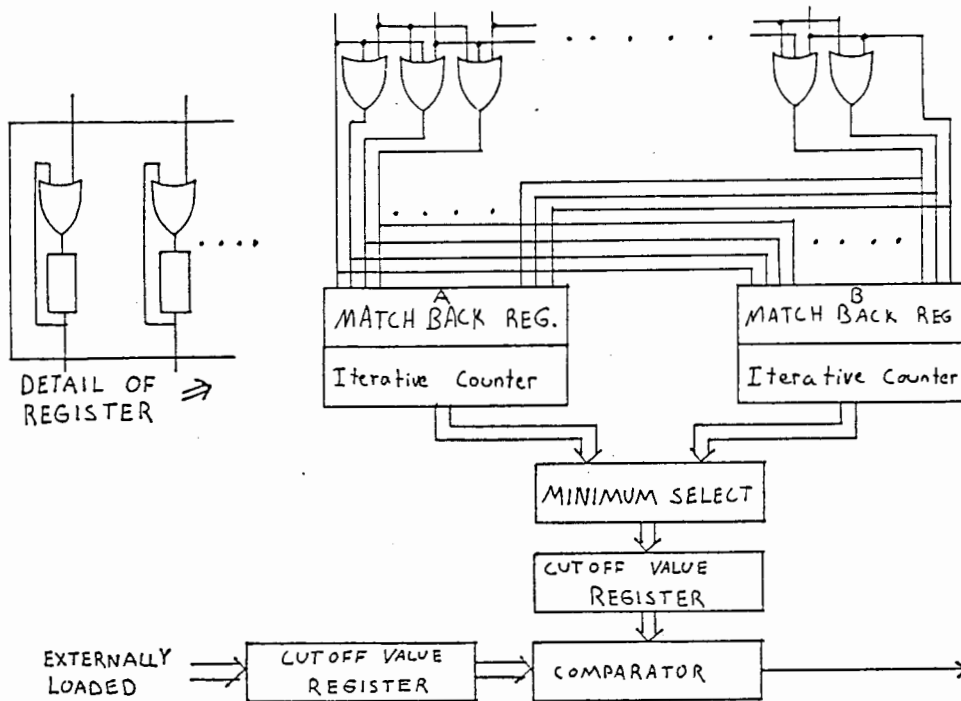


Figure 27.

Matchback Block

The cutoff value is used to screen out matches not meeting certain value. This section corresponds with the calculation of the match percentages of the algorithm. This prevents two codes which vary in size by a great deal but could still have all digits matched at the end of all comparisons. The cutoff value depends on the average length of codes being processed and is determined experimentally. The comparator compares the two cutoff value registers and controls the loading of the match value registers.

The match value register in figure 25 holds the match value. It waits for the results of the matchback percentages and if the percentages are high enough, the current match value is compared with the highest match value found previously. If the current match value is higher, it is loaded into the MV register and the library code index is loaded into the ID register.

When all the library codes have been compared with the unidentified code, the value in the ID register is looked at to see what plane type it represents and the unidentified code is now identified as being that plane type. After the plane has been identified, the value in the MV register is not important because the identification is made on a relative basis.

If the processor is being used in parallel with other processors as in figure 3, all the MV registers are compared and the highest one selected to identify the unidentified code. The procedure for parallel computation in this manner is straightforward. The library codes are divided into parts, with each part for one processor. The speedup possible is proportional to the number of processors. Theoretically, it would be possible to have one processor for each library code, although there would be overhead connected with selecting the match with the highest MV value.

CHAPTER VI

PERFORMANCE

Recognizing that comparisons and register operations on bits can be done in parallel on the hardware level, the complexity of the algorithm for comparison of two codes is $O(m)$, where m is the length of the longest code. More exactly, the complexity is $2 * (2m + mt)$, where the value of $2m$ is obtained from the m number of compare and shifts and the same number of operations on the match back registers. The value t is the number of cycles for the propagation delay of the submatch analysis block. The value obtained up to that point must be multiplied by two because of the requirement that the codes be cut in half and joined at the ends as was discussed in chapter II. Overhead steps have been left out of the complexity calculation, these are operations such as the initial loading of the registers.

A simple analysis of the hardware for one processor gives a more exact estimation of the speed as shown below. The assumption is $t = 3$ cycles:

Cycles

- | | |
|---|--|
| 3 | Comparing, matching back, and shifting |
| 3 | Analyzing results, (60% to 80% of the time
this step skipped) |

6	Total cycles for each number in code
---	--------------------------------------

The analysis step can be skipped some times because all the bits in R will be zero. This is easily checked with a small network of NOR gates. On the worst case assumption that the longest code is 64 digits long and the analysis step must be executed 40% of the time, the following calculation can be made:

$(6 * .40 * 64) + (3 * 64) = 346$ cycles for each library code. Assuming 64 cycles of overhead for loading, the total number of cycles required for each library code is 410 cycles. For the library used in the simulation and a cycle time of 2.5×10^7 secs, the time required to classify an object is approximately 0.0164 seconds.

CONCLUSION

In this thesis a new parallel algorithm for approximate strings matching has been developed. AN algorithm for object identification was presented and applied to the problem of aircraft recognition using INSM codes. With this system, it is possible to properly identify images very quickly even though they contain large amounts of errors or occlusions. The author knows of no other algorithms using INSM codes that can do this task so quickly and accurately. It also makes the use of INSM codes practical for many applications, because it removes the limitation that codes must match exactly.

The problems of the algorithm were discussed showing how the algorithm can be inaccurate under certain conditions. Experimental results were presented showing the algorithm works properly. While the simulation did return incorrect identifications in some cases, it is important to note there is no accurate, reliable way to indicate similarity between two images. When analyzing images, different factors can be assigned different values and it is difficult to gauge these values.

The approximate string matching algorithm has been shown to be fast due to its parallel nature. It also has

been shown to be reasonable accurate for a particular set of data. It can be expected that the algorithm could be applied to other types of data and obtain accurate results.

A corresponding processor design has been presented. While not completely specified and designed, the proposed hardware provides a suitable architecture for the implementation of the parallel algorithm developed here. Part of the processor has been designed as a VLSI chip by another student. The A and B registers, the EXNOR-AND comparison network and the R register have been laid out. However, the entire chip has not been completed.

The work in this thesis could be further applied to other problems such as acoustic analysis, geological data analysis and other problems using strings. The problem of longest common subsequence identification could also benefit from application of the algorithm developed here.

REFERENCES

1. Hall, P.A.V., Dowling, G.R., "Approximate String Matching," Computing Surveys, Vol. 12, No. 4, pp381-402.
2. Sankoff, D., Kruskal, J., eds., "Time Warps, String Edits, and Macromolecules: The Theory and Practice of Sequence Comparison," Addison-Wesley Pub., Reading Ma, 1983, 0-201-07809.
3. Anderson, J., "Cognitive Psychology and Its Implications." W.H. Freeman and Company, 1980, 1985.
4. Turney, J.L., Mudge, T.N., "Recognizing Partially Hidden Objects," SPIE Intelligent Robots and Computer Vision, 1984, Vol. 521, pp108-113.
5. Bhanu, B., Faugeras, O., "Shape Matching of Two-Dimensional Objects," IEEE Pattern Analysis and Image Processing, Vol 6, No. 2, pp137-155.
6. Duda, R., Hart, P., "Use of the Hough Transform to Detect Lines and Curves in Pictures," CACM, Vol 15, No. 1, pp 11-15.

7. Wallace, T.P., Wintz, P.A., "An efficient three-dimensional Aircraft Recognition Algorithm using Normalized Fourier descriptors," Computer Graphics Image Processing, Vol. 13, 1980, pp99-126.
8. Fu, K.S., "A Dynamic programming approach to sequential Pattern Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 3, pp313-26.
9. Fu, K.S., "Error-correcting parsing for syntactic pattern recognition," Data-structures, computer graphics and pattern recognition, Klinger et al., eds. Academic Press, New York, 1976.
10. Lin, W.C., Fu, K.S., "A Syntactic Approach to 3 Dimensional Object Recognition," IEEE Trans. on Systems, Man, and Cybernetics, Vol. 16, No. 3, pp405-422.
11. Kashyap, R.L., Oommen, B.J., "A Geometrical Approach to Polygonal Dissimilarity and Shape Matching," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 4, No. 6, pp649-655.

12. Tsai, W.H., Fu, K.S., "Attributed grammar-A tool for combining syntactic and statistical pattern recognition," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-10, Dec. 1980.
13. You, K.C., Fu, K.S., "Distorted Shape Recognition using Attributed Grammars and error-correcting techniques," Computer Graphics Image Processing, Vol. 13, 1980, pp1-16.
14. Wagner, R.A., "On the Complexity of the Extended String-to-String Correction Problem," Seventh Annual ACM Symp. on Theory of Computing, 1975, pp218-223.
15. Wagner, R.A., Fischer, M., "The String-to-String Correction Problem," J. Association of Computing Machinery, Vol. 21, No. 1, 1974, pp168-173.
16. Masek, W.J., Paterson, M.S., "A faster algorithm computing string edit distances," J. Computers and Systems Science, Vol. 20, pp18-31.
17. Lowrance, R., Wagner, R.A., "An Extension of the String-to-String Correction Problem," J. Association of Computing Machinery, Vol. 22, 1975, pp177-183.

18. Cheng, H.D., Fu, K.S., "VLSI Architectures for String Matching and Pattern Matching," Pattern Recognition, Vol 20, No 1.
19. Damerau, F.J., "A technique for computer detection and correction of spelling errors," Communications of the ACM, Vol. 7, No. 3., pp 171-176.
20. Peterson, W.W., "Error-correcting codes," Wiley, New York, 1961.
21. Tsai, W.H., "Attributed String Matching with merging for Shape Recognition," Seventh Intl. Conf. on Pattern Recognition, 84CH2046-1, pp1162-4.
22. Badi'i, F., Peikari B., "Invariant Numerical Shape Modeling," Proc. IEEE Conf. on Computer Vision and Pattern Recognition, June 1983.
23. Badi'i, F., Majd F., "Parallel Classification of 3D Moving Objects," Portland State University, 1985.
24. Abu-Mostafa, Y.S., Psaltis D. "Recognitive Aspects of Moment Invariants." IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 6, No. 6, pp698-706".

25. Andrejko, G., "Systolic Systems for the longest common subsequence problem." Computers and Artificial Intelligence, Vol. 5, No. 3, p199.
26. Apostolico, A., "Improving the worst case Performance of the Junt-Szymansky Strategy for the Longest Common Subsequence." Information Processing, Vol. 23, No. 2, pp63-69.
27. Ayache, N., "A New Approach for the Recognition and Positioning of two Dimensional Objects," IEEE Trans. On Pattern Analysis and Machine Intelligence, Vol. 8, No. 1, pp44-54.
28. Bacon, M.D., "Data Transmission," Macdonalds and Jane's London, 1973.
29. Bamich, B., "A General moment-invariants/attributed graph method for three dimensional object recognition from a single image," IEEE Journal of Robotics and Automation, vol RA-2 no 1, p31-41.
30. Bamieh, B., "A General Moments-Invariants Attributed graph method for 3-Dimensional Object Recognition from a single Image. Intl. J. of Robotics, Vol. 2, No. 1, pp31-41.

31. Benarie, J., Meiri, A.Z., "3D Object Recognition by Optimal Matching Search of Multinary Relations Graphs." Computer Vision Graphics and Image Processing, Vol. 37, No.3, pp345-361.
32. Bergen, J.R., Julesz, B., "Rapid Discrimination of Visual Patterns." IEEE Trans. Systems, Man, and Cybernetics, Vol. 13, No. 5, pp857-863.
33. Bolles, R.C., Cain, R.A., "Recognizing and locating Partially Visible Objects: The Local-Feature-Focus Method." Robot Vision, Pugh A., editor, Springer Verlag Inc., NY, 1983, 0-387-12073-4.
34. Brown, M.B., "Object Identification from 2-D Images." Image Vis C, Vol. 3, No. 4, pp150.
35. Bunke, H., Syntactic Analysis of Noisy Input Strings with an Application to the Analysis of Heart-Volume Curves." Seventh Intl. Conf. on Pattern Recognition, 1984, 84CH2046-1, pp1162-4.

36. Chen, C.H., "Automatic recognition of underwater transient signals," Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing, 1985, 85CH2118-8, p1270-2.
37. Chiang, Y., Fu, K.S., "VLSI Architectures for Syntactic Pattern Recognition," Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp474-481, June 1983.
38. Connell, J.H., Brady, M., "Generating and Generalizing Models of Vision Objects," Artificial Intelligence, Vol. 31, No.2, pp159-183.
39. Davis, L.S., "Shape Matching using Relaxation Techniques," IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 1, 1979, pp60-72.
40. Davis, L.S., Roussopoulos, N., "Approximate Pattern Matching in a Pattern Database System," Information Systems, Vol. 5, No. 2, pp107-119.
41. Dubois, D., Prade, H., "Fuzzy Pattern Matching with Extended Capabilities: L Proximity Notions, Importance Assessment, Random Sets," Proceedings of the NAFIPS '86-1986 Conf. of the North American Fuzzy Information Processing Soc., pp125-39.

42. Dudani, S.A., Breeding, K.J., "Aircraft Identification by Moment Invariants," IEEE Trans. on Computers, Vol. C-26 No. 1.
43. Faloutsos, C., "Access Methods for text", Computing Surveys, Vol. 17, No. 1, pp49-74.
44. Faugeras, O.D., Hebert, M., "The Representation, Recognition, and Locating of 3-D Objects," Intl. J. Robotics, Vol. 5, No. 3, pp27-52.
45. Fichet, B., Guad, E., "On Euclidean Images of a Set endowed with a Preordonnance," J. Mathematical Psychology, Vol. 31, No.1, pp24-43.
46. Flick, T.E., "A combinatorial approach for classification of patterns with missing information and random orientation," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 4, p482-90.
47. Freeman, H., "Shape description via the use of critical points," IEEE Computer Soc. Conf. on Pattern Recognition Image Processing, 1977.

48. Freeman, H., "Computer processing of line-drawing images" Computing Surveys, Vol. 6, 1974.
49. Freeman, H., "On the Encoding of Arbitrary Geometric Configurations," IRE Trans. on Electronic Computers, Vol. EC-10, No. 2, pp260-268.
50. Fukushima, K., Miyake, S., "Neocognition: A new Algorithm for pattern recognition tolerant of deformations and shifts in Position," Pattern Recognition, Vol. 15, No. 6, pp455-470.
51. Galil, Z., "Improved string matching with K mismatches," SIGACT News, Vol. 17, No. 4, p52-4.
52. Hakalahti, H., Harwood, D., "Two-Dimensional object recognition by matching local properties of contour points," Pattern Recognition Letters, Vol. 2, No. 4, pp235-238.
53. Hirschberg, D.S., "Algorithms for the longest common subsequence problem," J. Assoc. Computing Machinery, Vol. 24, No. 4, pp664-675.
54. Hopkins, R.H., Campbell, K.B., "Representations of Perceived Relations among the properties and Variables

of a Complex System," IEEE Trans. Systems, Man, and Cybernetics, Vol. 17, No.1, pp52-60.

55. Hsu, W.J., "Computing a Longest Common Subsequence for a set of Strings," BIT, Vol. 24, No. 1, pp45-59.
56. Hu, M.K., "Visual Pattern Recognition by Moment Invariants," IRE Trans. on Information Theory, Vol. IT-8, Feb. 1962.
57. Hwang, K., "VLSI architectures for Feature Extraction and Pattern Classification," Computer Vision Graphics and Image Processing, Vol. 24, No. 2, pp200-214.
58. Ito, T., "Hierarchical file Organization and its Application to Similar String Matching," ACM Trans. on Database Systems, Vol. 8, No. 3, pp410-433.
59. Jacobsen, L., Wechsler, H., "Invariant Image Representation: A path toward solving the Bin-Picking problem," Proc. IEEE Intl. Conf. on Robotics, pp189-199, Mar. 1984.
60. Johnson, J.H., "Formal Models for String Similarity." Doctoral Thesis, University of Waterloo, Waterloo Ontario, Canada 1983.

61. Kandel, A., "Fuzzy Techniques in Pattern Recognition," John Wiley & Sons, Inc., NY 1982, 0-471-091336-7.
62. Kandel, A., "Algorithmic Approach to Fuzzy Classification," Proc. of Fourteenth Intl. Symp. on Multiple Valued Logic, 1984, 84CH2052-9, pp142.
63. Kashyap, R.L., Oommen, B.J., "An effective algorithm for string correction using generalized edit distances-I. Description of the algorithm and some applications," Information Science, Vol. 23, 1981, pp123-142.
64. Keller, J.M., "Membership Function Issues in Fuzzy Pattern Recognition," 1985 Proceedings of the International Conference on Cybernetics and Society, 85CH2253-3, p210-14.
65. Knoll, T.F., Jain, R.C., "Recognizing Partially Visible Objects using Feature Indexed Hypothesis," Intl. J. Robotics, Vol. 2, No. 1, pp3-13.
66. Korn, M.R., Dyer, C.R., "3D Multiview Object Representation for Model-Based Object Recognition," Pattern Recognition, Vol. 20, No.1, pp91-103.

67. Landau, G.M., "Efficient String matching in the presence of errors," Symposium on Foundations of Computer Science, 26th Annual, 1985, 85CH2224-4, pl26-36.
68. Lowe, D.G., "3-Dimensional Object Recognition from Single Two-Dimensional Images," Artificial Intelligence, Vol. 31, No. 3, pp355-395.
69. Lu, S.Y., Fu, K.S., "A sentence-to-sentence clustering procedure for pattern analysis," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-8(5), pp381-389.
70. McDonald, R.P., "Describing the Elephant Structure and Function in Multivariate Data," Psychometrika, Vol. 51, No. 4, pp513-534.
71. Mowforth P., "An Approach to Object Recognition with Parallel Computation," Computers and Artificial Intelligence, Vol. 2, No. 6, pp531-540.
72. Mudge, T.N., Abdel-Rahman, T.S., "Case Study of a program for the recognition of occluded parts," IEEE Workshop on Computer Architecture for Pattern Analysis and Image Database Management, 1983, 83CM1929-9.

73. Mukhopadhyay, A., "Hardware Algorithms for String Processing," IEEE Computer, 1980, pp508-511.
74. Mukhopadhyay, A., "A fast algorithm for the longest-common-subsequence problem," Information Science., Vol. 20, pp69-82.
75. Nakatsy, N., "A Longest Common Subsequence algorithm suitable for Similar text-strings," Acta Inf., Vol. 18, No. 2, pp171-179.
76. Nesbit, J.C., "Approximate String Matching in Response Analysis," J. Computer Based Instruction, Vol. 12, No. 3, pp71-75.
77. Nesbit, J.C., "The Accuracy of Approximate String Matching Algorithms," J. Computer Based Instruction, Vol. 13, No. 3, pp80-83.
78. Neveu, C.F., Dyer, C.R., "Two Dimensional Object Recognition using Multi Resolution Models," Computer Vision Graphics and Image Processing, Vol. 34, No.1, pp52-65.

79. Oommen, B.J., "Pattern Recognition with strings, substrings and boundaries," Purdue Univ., West Lafayette, IN. 1982, UMI-DA8225750.
80. Ozaki, H., Waku, S., Mohri, A., "Pattern Recognition of a grasped object by unit vector distribution," IEEE Trans. System, Man, and Cybernetics, Vol. 12, No. 3, pp315-324.
81. Pavlidis, T., "A review of algorithms for shape analysis," Computer Graphics Image Processing, Vol. 7, 1978.
82. Peltola, H., "Algorithms for some string matching problems arising in molecular genetics," Information Processing 1983, Proceedings of the IFIP 9th World Computer Congress, 0-444-86729-5, pp59-64.
83. Perkins, W.A., "A model-based vision system for industrial parts," IEEE Trans. on Computers, Vol. C-27, Feb. 1978.
84. Ronacher, B., Bautz, W., "Human Pattern Recognition: Individually different strategies in Analyzing complex stimuli," Biological Cybernetics, Vol. 51, No. 4, pp249-263.

85. Ronancher, B., "Human Pattern Recognition: Evidence for a switching between strategies in analyzing complex stimuli," Biological Cybernetics, Vol. 51, No. 3, pp205-210.
86. Rutkowski, W.S., "Recognition of Occluded Shapes using Relaxation," Computer Vision Graphics and Image Processing, Vol. 19, No. 2, pp111-128.
87. Sellers, P.H., "An algorithm for the distance between two finite sequences," J. Combinatorial Theory, Ser. A16, pp253-258.
88. Shima, Y., "A fast Algorithm for template pattern matching based on probability of occurrence of subpatterns," Systems and Computers, Japan, Vol. 16, No. 6, p87-95.
89. Silberberg, T.M., Harwood, D.A., "Object Recognition using Oriented Model Points," Computer Vision Graphics and Image Processing, Vol. 35, No.1, pp47-71.
90. Smith, G.D.U., "A Comparison of three string matching algorithms," Software Pract. Exper., Vol. 12, No. 1, pp57-66.

91. Steele, J.M., "Long Common Subsequences and the proximity of two random strings," SIAM J. Appl. Mathematics, Vol. 42, No. 4, pp731-737.
92. Stimets, R.W., Ying, W.L., "Rapid Recognition of Object Outlines in Reduced Resolution Images," Pattern Recognition, Vol. 19, No.1, pp21-33.
93. Takahashi, K., Yamada, H., "A New String Search Hardware Architecture for VLSI, 13th Annual Intl. Symp. on Computer Architecture, 1986, 86CH2291-3, pp20-27.
94. Tichy, W.F., "The String-to-String Correction Problem with Block Moves," ACM trans. on Computer Systems, Vol. 2, No. 4, pp280-308.
95. Tsui, H.T., Chan, M.H., "Recognition of Partial Occluded Two-Dimensional Objects," IEE Proceedings-E: Computers and Digital Techniques, Vol. 134, No. 1, pp9-16.
96. Turney, J., Mudge, T., "Recognizing Partially Occluded Parts," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 7, No. 4.

97. Ukkonen, E., "On Approximate String Matching," Foundations of Computational Theory, Proceedings of the 1983 Intl. FCT Conf., Springer Verlag, 0-38712-6899.
98. Ukkonen, E., "Finding Approximate Patterns in Strings," J. Algorithms, Vol. 6, No. 1, pp132-137.
99. Ukkonen, E., "Algorithms for Approximate String Matching," Information and Control, Vol. 64, No.1-3, pp100-118.
100. Ullman, J.R., "An Investigation of occlusion in one dimension," Computer Vision Graphics and Image Processing, Vol. 22, No.1, pp194-203.
101. Vernon, D., "Two Dimensional Object Recognition using Partial Contours," Image and Vision Computing, Vol. 5, No. 1, pp21-27.
102. Vogels, R., Orban, G.A., "Decision Processes in Visual-Discrimination of Line Orientation," J. Experimental Psychology-Human Perception and Performance, Vol. 12, No.2, pp115-132.

103. Williams, T., "Chip finds degree of similarity between strings," Computer Design, Vol 23, No. 6, pp204.
104. Yang, Y.H., Sze, T.W., "A Simple Contour Matching Algorithm," Proc. IEEE Conf. Pattern Recognition Image Processing, 1981, pp562-564.

APPENDIX 1

DETAILED ALGORITHM

$A(x,0)$, $B(x,0)$ - array into which codes are loaded
 $A(x,1)$, $B(x,1)$ - array which indicates matched numbers
 R - array containing results of comparisons
 $TOP_OF_LIBRARY$ - number of codes in the image code library.
 n - length of the longer code
 m - length of the shorter code
 LC - longer code
 SC - shorter code
 A_1, A_2, B_1, B_2 - values related to the percentage of numbers matched
 $A_MATCHBACK_VAL$ - matchback values
 $B_MATCHBACK_VAL$ - matchback values
 $N(1), N(2)$ - values of the numbers matched in codes
 $IDENTIFIED_VALUE$ - The highest current match value (MV)
 $IDENT$ - indicates the plane identified

Note: In the following algorithm, the unidentified code and each library code are compared separately. Code length determines where codes are placed.

```

IDENTIFIED_VALUE = 0
IDENT = 0
for INDEX = 1 to TOP_OF_LIBRARY do
  SUB <- 0
  MV <- 0
  A <- larger code,
  A(1,0) <- LC1, A(2,0) <- LC2, ..., A(n,0) <- LCn
  B <- shorter code,
  B(1,0) <- SC1, B(2,0) <- SC2, ..., B(m,0) <- SCm
  for i = 1 to 2 do
    A(p,1) <- 0      p = 1, 2, ..., n
    B(p,1) <- 0      p = 1, 2, ..., m

  for j = 1 to n-1 do
    Rk <- [A(k,0) ⊗ B(k,0)] * [A(k+1,0) ⊗ B(k+1,0)] *
           [A(k+2,0) ⊗ B(k+2,0)]      k = 1, 2, ..., m-2

    A(k,1) <- Rk + A(k,1)
    A(k+1,1) <- Rk + A(k+1,1)
    A(k+2,1) <- Rk + A(k+2,1)

    B(k,1) <- Rk + B(k,1)
    B(k+1,1) <- Rk + B(k+1,1)
    B(k+2,1) <- Rk + B(k+2,1)
  
```

Circular right shift both elements of A

```

for l = 1 to m-2 do
  if Rl = 1 then
    SUB = SUB + 1
  else
    MV = MV + SUB3
    SUB = 0

```

END for

A_MATCHBACK_VAL = NUMBER_OF_ONES(A(x,l))

B_MATCHBACK_VAL = NUMBER_OF_ONES(B(x,l))

N(i) = min(A_MATCHBACK_VAL, B_MATCHBACK_VAL)

A <- longer code

```

A(1,0) <- LC(n/2)+1, A(2,0) <- LC(n/2)+2, .....
A(n/2,0) <- LC1, ..... A(n,0) <- LC(n/2)

```

B <- longer code

```

B(1,0) <- SC(n/2)+1, B(2,0) <- SC(n/2)+2, .....
B(n/2,0) <- SC1, ..... B(n,0) <- SC(n/2)

```

END for

A1 = N(1) / n

B1 = N(1) / m

A2 = N(2) / n

B2 = N(2) / m

```

if (A1 > 65% * B1 > 65%) OR (A2 > 65% * B2 > 65%) then
  if IDENTIFIED_VALUE < MV then
    IDENTIFIED_VALUE = MV
    IDENT = INDEX

```

END ALGORITHM

After the end of the algorithm, IDENT will contain the index of the identified plane from the library. Also the 65% in the last step can be changed depending on the average lengths of the library codes.

APPENDIX 2
ALGORITHM SIMULATION PROGRAM

```

#include "defs.h"
void printres(str)
int *str;
{
    int i=0;
    char c;
    while(i<=wlen)
    {
        printf("%d",str[i]);
        i++;
    }
    printf(" \n");
}
void printar(str)
int *str;
{
    int i=0;
    char c;
    while((str[i]!=99) && (i<wlen))
    {
        printf("%c",str[i]);
        i++;
    }
    printf(" \n");
}
void printreg2()
{
    int i=0;
    for(i=0;i<=wlen;i++)
    {
        printf("%c",r2[i].val);
    }
    printf("\n");
    for(i=0;i<=wlen;i++)
        printf("%d",r2[i].dc);
    printf("\n");
    for(i=0;i<=wlen;i++)
        printf("%d",r2[i].mt);
    printf("\n");
}
void printreg()
{
    int i=0;
    for(i=0;i<=wlen;i++)
    {
        printf("%c",r1[i].val);
    }
    printf("\n");
    for(i=0;i<=wlen;i++)
        printf("%d",r1[i].dc);
    printf("\n");
    for(i=0;i<=wlen;i++)

```

```

    printf("%d",rl[i].mt);
    printf("\n");
}
int readlib()
{
    int count = 0;
    FILE *libfile;
    char c;
    int pos = 0;
    libfile=fopen("codes","r");
    while((feof(libfile)) == 0)
    {
        for(pos=0;pos<=wlen;pos++)
            lib[count][pos]=99;
        pos=0;
        do
        {
            c=(char) (fgetc(libfile));
            if((c!='\n') && (c!=EOF))
            {
                lib[count][pos]=c;
                pos++;
            }
        } while((c!='\n') && (c!=EOF));
        count++;
    }
    libsize=count--;
    return 1;
}
int readin(in)
int *in;
{
    int pos;
    int c;
    for(pos=0;pos<=wlen;pos++)
        in[pos]=99;
    pos=0;
    if((feof(stdin)) == EOF)
        return 1;
    else
    {
        do
        {
            c=getchar();
            if((c!='\n') && (c!=EOF))
            {
                in[pos]=c;
                pos++;
            }
        } while((c!='\n') && (c!=EOF));
    }
    if(c==EOF)

```



```

(r1[0].val==r2[0].val))
    resreg[i]=1;
    }
    }
    else
    {
if((r1[i].dc!=1)&&(r2[i].dc!=1)&&(r1[i+1].dc!=1)&&(r2[i+1].dc!=1)&&(r1[i+2].dc!=1)&&(r2[i+2].dc!=1))
    {
        if((r1[i].val==r2[i].val) &&
(r1[i+1].val==r2[i+1].val) &&(r1[i+2].val==r2[i+2].val))
            resreg[i]=1;
    }
    }
}
void clearregs()
{
    int pos;
    for(pos=0;pos<wlen;pos++)
    {
        r1[pos].val=99;
        r1[pos].dc=1;
        r1[pos].mt=0;
        r2[pos].val=99;
        r2[pos].dc=1;
        r2[pos].mt=0;
        resreg[pos]=0;
    }
}
void spshift(str)
int *str;
{
    int t;
    int i;
    int v1;
    int v2;
    int p;
    t=mylen(str);
    for(p=0;p<t/2;p++)
    {
        v1=str[t-1];
        for(i=0;i<t;i++)
        {
            v2=str[i];
            str[i]=v1;
            v1=v2;
        }
    }
}
void shiftr1(len)

```

```

int len;
{
    int pos,tlv,tld,t2v,t2d,tlm,t2m;
    tlv=r1[len-1].val;
    tld=r1[len-1].dc;
    tlm=r1[len-1].mt;
    for(pos=0;pos<=len-1;pos++)
    {
        t2v=r1[pos].val;
        t2d=r1[pos].dc;
        t2m=r1[pos].mt;
        r1[pos].val=tlv;
        r1[pos].dc=tld;
        r1[pos].mt=tlm;
        tlv=t2v;
        tld=t2d;
        tlm=t2m;
    }
}
void mshiftrl(len)
int len;
{
    int pos;
    int tlv,tld,t2v,t2d,tlm,t2m;
    tlv=99;
    tld=1;
    tlm=0;
    for(pos=0;pos<=wlen-1;pos++)
    {
        t2v=r1[pos].val;
        t2d=r1[pos].dc;
        t2m=r1[pos].mt;
        r1[pos].val=tlv;
        r1[pos].dc=tld;
        r1[pos].mt=tlm;
        tlv=t2v;
        tld=t2d;
        tlm=t2m;
    }
}
int mylen(str)
int *str;
{
    int val;
    int pos;
    val=0;
    pos=0;
    while((str[pos]!=99) && (pos<wlen))
    {
        val++;
        pos++;
    }
}

```



```

    return val;
}
int analyzereg()
{
    int pos;
    int count;
    int tempans;
    int start;
    int ans;
    tempans=0;
    ans=0;
    start=0;
    pos=0;
    if((resreg[pos]==0) && (pos < wlen))
        pos++;
    start=pos;
    for(count=0;count<wlen;count++)
    {
        if(resreg[pos]==1)
            tempans++;
        else
        {
            if(tempans!=0)
            {
                tempans=tempans + 2;
                ans=ans+(tempans*tempans*tempans);
                tempans=0;
            }
        }
        pos++;
    }
    return ans;
}
int match(t1,t2,len)
int *t1;
int *t2;
int len;
{
    int pos;
    int temp;
    int trash[wlen];
    int *in1=trash;
    int *in2=trash;
    int rotate;
    int count;
    int templ;
    int intres;
    int ans;
    int pos2;
    in2=t2;
    in1=t1;
    ans=0;

```

```

pos=0;
temp=0;
intres=0;
for(rotate=0;rotate<2;rotate++)
{
    clearregs();
    pos=0;
    while((in2[pos]!=99) && (pos<wlen))
    {
        r1[pos].val=in2[pos];
        r1[pos].dc=0;
        pos++;
    }
    pos2=0;
    pos=0;
    while((in1[pos]!=99) && (pos<wlen))
    {
        r2[pos2].val=in1[pos];
        r2[pos2].dc=0;
        pos++;
        pos2++;
    }
    for(count=0;count<len;count++)
    {
        comp(wlen);
        for(pos=0;pos<wlen;pos++)
        { /* use results in resreg to blank out parts
of r1 and r2 */
            if(resreg[pos]==1)
            {
                r1[pos].mt=1;
                r2[pos].mt=1;
                if(pos==wlen)
                {
                    r1[0].mt=1;
                    r2[0].mt=1;
                }
            }
            else
            {
                r1[pos+1].mt=1;
                r2[pos+1].mt=1;
                r1[pos+2].mt=1;
                r2[pos+2].mt=1;
            }
        }
    }
    ans=ans+analyzereg();
    intres=calc();
    temp=temp+intres;
    shiftrl(len);
    for(pos=0;pos<=wlen;pos++)
        resreg[pos]=0;
}

```

```

    }
    if(rotate==0)
    {
        v1=regcalc();
    }
    else
    {
        templ=regcalc();
        v2=templ;
    }
    spshift(in1);
    spshift(in2);
}
return ans;
}
main()
{
    int mcount;
    int len1,len2;
    int v;
    int vrlp;
    int vr2p;
    int vrl;
    int vr2;
    int incom[wlen];
    int tlib;
    if(readlib()==0)
        return 1;
    tlib=libsize;
    v1=0;
    v2=0;
    while((readin(incom))!=1)
    {
        /* printf("just before countinc, libsize=%d\n",tlib); */
        printar(incom);
        for(mcount=0;mcount<tlib-1;mcount++)
        {
            /* printf("just before clearregs, mcount=%d
lib=%d\n",mcount,tlib); */
            clearregs();
            len1=mylen(lib[mcount]);
            len2=mylen(incom);
            if(len1<len2)
            {
                clearregs();
                v=match(lib[mcount],incom,len2);
            }
            else
            {
                clearregs();
                v=match(incom,lib[mcount],len1);
            }

```

```

vr1p=(v1*100)/len2;
vr2p=(v1*100)/len1;
vr1=(v2*100)/len2;
vr2=(v2*100)/len1;
    if(((vr1p>65) && (vr2p>65)) | ((vr1>65) && (vr2>65)))
    {
        printf("anreg = %d ",v);
        printf("r = %d and %d, %d %d, mc=%d to =%d %d\n",vr1p,vr2p,vr1,vr2,mcount,vr1p+vr2p,vr1+vr2);
    }
}
}

```

APPENDIX 3
NUMERIC DATA RETURNED BY SIMULATION

PLANE MODEL 2

72642614613672634616

MV = 850 Plane code = 0
 MV = 587 Plane code = 1
 MV = 1918 Plane code = 2
 MV = 540 Plane code = 3
 MV = 573 Plane code = 17
 MV = 755 Plane code = 27
 MV = 873 Plane code = 28
 MV = 570 Plane code = 30
 MV = 742 Plane code = 39
 MV = 804 Plane code = 46
 MV = 813 Plane code = 86
 MV = 864 Plane code = 116
 High MV is 2 Plane class is
 MIG

7263463424361346136

MV = 1854 Plane code = 0
 MV = 891 Plane code = 1
 MV = 1227 Plane code = 2
 MV = 705 Plane code = 8
 MV = 621 Plane code = 10
 MV = 1324 Plane code = 15
 MV = 533 Plane code = 17
 MV = 1621 Plane code = 27
 MV = 2175 Plane code = 28
 MV = 621 Plane code = 29
 MV = 884 Plane code = 30
 MV = 1121 Plane code = 35
 MV = 1827 Plane code = 39
 MV = 888 Plane code = 40
 MV = 499 Plane code = 41
 MV = 496 Plane code = 65
 MV = 928 Plane code = 86
 MV = 871 Plane code = 115
 MV = 1073 Plane code = 116
 MV = 658 Plane code = 126
 High MV is 28 Plane class
 is MIRAGE

726424361346136724134616

MV = 2491 Plane code = 0
 MV = 691 Plane code = 4
 MV = 2659 Plane code = 28
 MV = 1086 Plane code = 30
 MV = 2555 Plane code = 33
 MV = 1016 Plane code = 35
 MV = 2636 Plane code = 39
 MV = 1218 Plane code = 45
 MV = 739 Plane code = 86
 MV = 869 Plane code = 120
 MV = 1565 Plane code = 122

High MV is 28 Plane class
is MIRAGE

PLANE MODEL 45

66116726614134613431613

MV = 1595 Plane code = 0
 MV = 614 Plane code = 17
 MV = 1806 Plane code = 34
 MV = 1887 Plane code = 39
 MV = 7624 Plane code = 45
 MV = 732 Plane code = 90
 MV = 962 Plane code = 105
 MV = 793 Plane code = 116
 MV = 1156 Plane code = 122
 High MV is 45 Plane class
 is MIRAGE

66136141334613431613

MV = 1154 Plane code = 0
 MV = 678 Plane code = 15
 MV = 1225 Plane code = 27
 MV = 3090 Plane code = 33
 MV = 1738 Plane code = 34
 MV = 708 Plane code = 35
 MV = 1275 Plane code = 39
 MV = 1241 Plane code = 40
 MV = 6949 Plane code = 45
 MV = 1206 Plane code = 46
 MV = 785 Plane code = 47
 MV = 1308 Plane code = 50
 MV = 1033 Plane code = 58
 MV = 1190 Plane code = 98
 MV = 810 Plane code = 105
 MV = 702 Plane code = 121
 MV = 516 Plane code = 134
 High MV is 45 Plane class
 is MIRAGE

6611672641334613431613

MV = 958 Plane code = 0
 MV = 648 Plane code = 4
 MV = 560 Plane code = 17
 MV = 2874 Plane code = 33
 MV = 1063 Plane code = 39
 MV = 3828 Plane code = 45
 MV = 634 Plane code = 90
 MV = 763 Plane code = 105
 MV = 584 Plane code = 121
 MV = 850 Plane code = 122
 High MV is 45 Plane class
 is MIRAGE

6613614134613341431613

MV = 1264 Plane code = 0

```

MV = 1597    Plane code = 27    is Phantom
MV = 1662    Plane code = 33    726616134334
MV = 1798    Plane code = 34    MV = 398    Plane code = 100
MV = 1338    Plane code = 39    MV = 432    Plane code = 108
MV = 6720    Plane code = 45    High MV is 108 Plane class
MV = 1089    Plane code = 47    is Phantom
MV = 2026    Plane code = 50    PLANE MODEL 2
MV = 877     Plane code = 56
MV = 749     Plane code = 58    726426146136724134616
MV = 705     Plane code = 91    MV = 1530    Plane code = 0
MV = 658     Plane code = 105   MV = 671     Plane code = 1
MV = 722     Plane code = 121   MV = 4528    Plane code = 2
MV = 621     Plane code = 134   MV = 617     Plane code = 4
High MV is 45 Plane class
is MIRAGE
66116726614134613341431613
MV = 1221    Plane code = 34    MV = 563     Plane code = 17
MV = 2261    Plane code = 45    MV = 1462    Plane code = 27
MV = 695     Plane code = 56    MV = 1224    Plane code = 28
MV = 1136    Plane code = 122   MV = 479     Plane code = 29
High MV is 45 Plane class
is MIRAGE
66136141334613341431613
MV = 1154    Plane code = 33    MV = 873     Plane code = 30
MV = 1265    Plane code = 34    MV = 462     Plane code = 35
MV = 4500    Plane code = 45    MV = 1170    Plane code = 39
MV = 1577    Plane code = 50    MV = 984     Plane code = 40
MV = 668     Plane code = 58    MV = 746     Plane code = 46
MV = 587     Plane code = 134   MV = 796     Plane code = 86
High MV is 45 Plane class
is MIRAGE
661167266141334613341431613
MV = 1755    Plane code = 45    MV = 503     Plane code = 87
MV = 994     Plane code = 122   MV = 577     Plane code = 95
MV = 560     Plane code = 134   MV = 1110    Plane code = 116
High MV is 45 Plane class
is MIRAGE
72413463424361346136
MV = 2215    Plane code = 0
MV = 4502    Plane code = 2
MV = 637     Plane code = 4
MV = 415     Plane code = 10
MV = 1442    Plane code = 15
MV = 584     Plane code = 17
MV = 2100    Plane code = 27
MV = 2515    Plane code = 28
MV = 766     Plane code = 29
MV = 1086    Plane code = 30
MV = 2306    Plane code = 33
MV = 1347    Plane code = 35
MV = 2306    Plane code = 39
MV = 1030    Plane code = 40
MV = 688     Plane code = 41
MV = 1093    Plane code = 45
MV = 560     Plane code = 65
MV = 857     Plane code = 86
MV = 1144    Plane code = 98
MV = 1067    Plane code = 115
MV = 1232    Plane code = 116
MV = 839     Plane code = 120
PLANE MODEL 53
61343311
MV = 434     Plane code = 53
MV = 145     Plane code = 73
MV = 162     Plane code = 79
MV = 1024    Plane code = 141
High MV is 141 Plane class
is F105
7266436134334
MV = 233     Plane code = 93
MV = 530     Plane code = 100
MV = 469     Plane code = 108
High MV is 100 Plane class

```

MV = 766 Plane code = 121
 MV = 648 Plane code = 126
 High MV is 2 Plane class is
 MIG

7263463426146136

MV = 648 Plane code = 0
 MV = 1181 Plane code = 1
 MV = 3376 Plane code = 2
 MV = 1262 Plane code = 3
 MV = 1026 Plane code = 7
 MV = 560 Plane code = 8
 MV = 435 Plane code = 11
 MV = 665 Plane code = 15
 MV = 354 Plane code = 17
 MV = 692 Plane code = 27
 MV = 395 Plane code = 29
 MV = 567 Plane code = 35
 MV = 648 Plane code = 39
 MV = 574 Plane code = 40
 MV = 546 Plane code = 41
 MV = 675 Plane code = 46
 MV = 317 Plane code = 65
 MV = 439 Plane code = 85
 MV = 661 Plane code = 88
 MV = 476 Plane code = 96
 MV = 769 Plane code = 116
 MV = 520 Plane code = 118
 MV = 1080 Plane code = 119
 MV = 496 Plane code = 126
 MV = 520 Plane code = 128
 High MV is 2 Plane class is
 MIG

72642436134613672634616

MV = 1769 Plane code = 0
 MV = 2095 Plane code = 28
 MV = 901 Plane code = 30
 MV = 2083 Plane code = 39
 MV = 739 Plane code = 86
 MV = 1674 Plane code = 122
 High MV is 28 Plane class
 is MIRAGE

PLANE MODEL 71

734341263151614

MV = 1322 Plane code = 71
 High MV is 71 Plane class
 is B57
 73434351263414
 MV = 438 Plane code = 16
 MV = 418 Plane code = 20
 MV = 1907 Plane code = 71

MV = 749 Plane code = 78
 MV = 290 Plane code = 105
 High MV is 71 Plane class
 is B57

73434351263151614

High MV is 0 No
 identification possible

PLANE MODEL 17

726434311343461316672446

MV = 2412 Plane code = 13
 MV = 7669 Plane code = 17
 MV = 1022 Plane code = 28
 MV = 873 Plane code = 30
 MV = 604 Plane code = 120
 MV = 479 Plane code = 134
 High MV is 17 Plane class
 is MIG

72611341316672446

MV = 4799 Plane code = 17
 MV = 773 Plane code = 18
 MV = 344 Plane code = 21
 MV = 455 Plane code = 29
 MV = 401 Plane code = 35
 MV = 439 Plane code = 96
 High MV is 17 Plane class
 is MIG

726434311341316672446

MV = 853 Plane code = 13
 MV = 3541 Plane code = 17
 MV = 1135 Plane code = 18
 MV = 449 Plane code = 94
 High MV is 17 Plane class
 is MIG

46131634311343

MV = 479 Plane code = 8
 MV = 378 Plane code = 15
 MV = 590 Plane code = 16
 MV = 695 Plane code = 19
 MV = 492 Plane code = 20
 MV = 813 Plane code = 33
 MV = 731 Plane code = 40
 MV = 624 Plane code = 45
 MV = 307 Plane code = 50
 MV = 604 Plane code = 81
 MV = 425 Plane code = 90
 MV = 425 Plane code = 91
 MV = 910 Plane code = 98
 MV = 775 Plane code = 105
 High MV is 98 Plane class
 is Phantom

PLANE MODEL 35

724636134262661346346

MV = 857 Plane code = 2
 MV = 496 Plane code = 4
 MV = 778 Plane code = 9
 MV = 830 Plane code = 15
 MV = 1137 Plane code = 27
 MV = 2144 Plane code = 28
 MV = 3100 Plane code = 29
 MV = 1713 Plane code = 30
 MV = 6626 Plane code = 35
 MV = 533 Plane code = 40
 MV = 658 Plane code = 41
 MV = 772 Plane code = 45
 MV = 452 Plane code = 46
 MV = 1399 Plane code = 59
 MV = 766 Plane code = 65
 MV = 371 Plane code = 87
 MV = 1116 Plane code = 116
 High MV is 35 Plane class
 is MIRAGE

72463634161341436346

MV = 1252 Plane code = 10
 MV = 786 Plane code = 15
 MV = 685 Plane code = 16
 MV = 1278 Plane code = 29
 MV = 2099 Plane code = 35
 MV = 1207 Plane code = 86
 MV = 964 Plane code = 91
 MV = 778 Plane code = 116
 MV = 543 Plane code = 120
 High MV is 35 Plane class
 is MIRAGE

724636134262661341436346

MV = 587 Plane code = 4
 MV = 1830 Plane code = 28
 MV = 3550 Plane code = 29
 MV = 1719 Plane code = 30
 MV = 3543 Plane code = 35
 MV = 1644 Plane code = 59
 MV = 712 Plane code = 65
 MV = 957 Plane code = 86
 High MV is 29 Plane class
 is MIRAGE

6343461361453423461

MV = 547 Plane code = 2
 MV = 827 Plane code = 8
 MV = 408 Plane code = 9
 MV = 3406 Plane code = 40
 MV = 486 Plane code = 41
 MV = 937 Plane code = 46

MV = 371 Plane code = 65
 MV = 472 Plane code = 128
 High MV is 40 Plane class
 is MIRAGE

634346236413461

MV = 506 Plane code = 8
 MV = 1063 Plane code = 14
 MV = 830 Plane code = 15
 MV = 493 Plane code = 16
 MV = 449 Plane code = 20
 MV = 2269 Plane code = 40
 MV = 364 Plane code = 42
 MV = 445 Plane code = 91
 MV = 918 Plane code = 98
 MV = 1245 Plane code = 115
 MV = 428 Plane code = 121
 MV = 736 Plane code = 127
 MV = 648 Plane code = 128
 MV = 542 Plane code = 133
 High MV is 40 Plane class
 is MIRAGE

634346236413461

MV = 506 Plane code = 8
 MV = 1063 Plane code = 14
 MV = 830 Plane code = 15
 MV = 493 Plane code = 16
 MV = 449 Plane code = 20
 MV = 2269 Plane code = 40
 MV = 337 Plane code = 42
 MV = 445 Plane code = 91
 MV = 918 Plane code = 98
 MV = 1245 Plane code = 115
 MV = 428 Plane code = 121
 MV = 736 Plane code = 127
 MV = 648 Plane code = 128
 MV = 542 Plane code = 133
 High MV is 40 Plane class
 is MIRAGE

PLANE MODEL 77

73146334134242633441344

MV = 793 Plane code = 76
 MV = 8272 Plane code = 77
 High MV is 77 Plane class
 is B57

7314633534433441344

MV = 631 Plane code = 70
 MV = 5920 Plane code = 77
 High MV is 77 Plane class
 is B57
 7314633534242633441344

MV = 614 Plane code = 76
MV = 4645 Plane code = 77
High MV is 77 Plane class
is B57

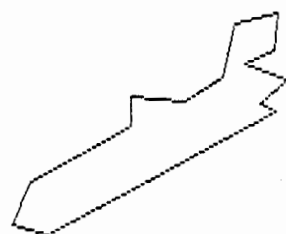
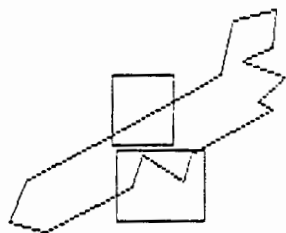
PLANE MODEL 82

44334343153343
MV = 806 Plane code = 81
MV = 1592 Plane code = 82
High MV is 82 Plane class
is B57
1443343431
MV = 604 Plane code = 81
MV = 1053 Plane code = 82
MV = 135 Plane code = 131
High MV is 82 Plane class
is B57
1443343431533
MV = 388 Plane code = 51
MV = 1044 Plane code = 82
High MV is 82 Plane class
is B57

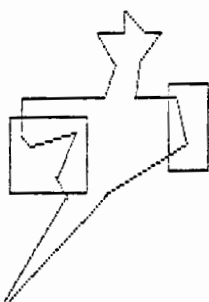
PLANE MODEL 62

7261422631516
MV = 918 Plane code = 62
High MV is 62 Plane class
is B57
7261672642263
MV = 371 Plane code = 62
High MV is 62 Plane class
is B57
72616726422631516
High MV is 0 No
identification possible

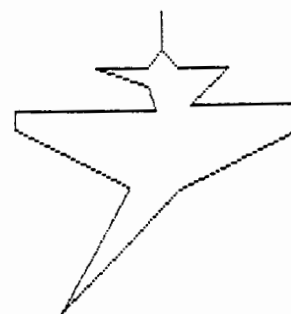
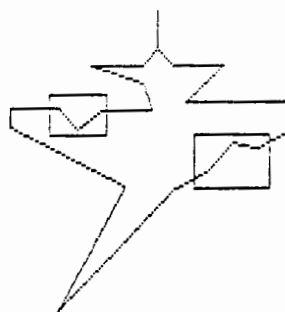
APPENDIX 4
PICTURES OF PLANES USED
DIRECT ENTRY AND COMPUTER PROCESSED



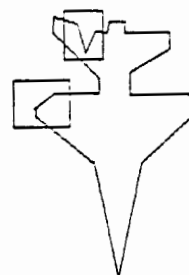
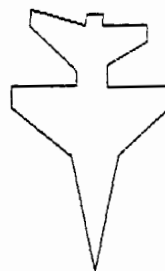
PLANE 17



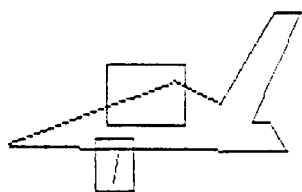
PLANE 40



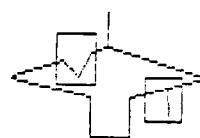
PLANE 35



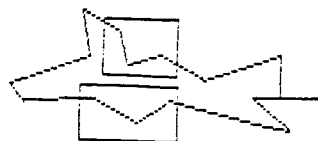
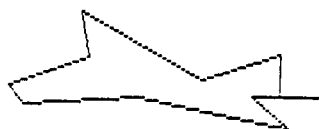
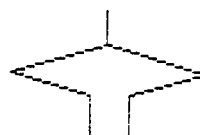
PLANE 45



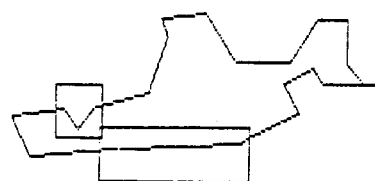
PLANE 53



PLANE 62



PLANE 71



PLANE 77